

PGMcpp: PRIMED Grid Modelling (in C++)

Generated by Doxygen 1.9.1



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Combustion Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Combustion()	8
4.1.2.2 ~Combustion()	9
4.2 Controller Class Reference	9
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	9
4.2.2.1 Controller()	9
4.2.2.2 ~Controller()	10
4.3 Diesel Class Reference	10
4.3.1 Detailed Description	11
4.3.2 Constructor & Destructor Documentation	11
4.3.2.1 Diesel()	11
4.3.2.2 ~Diesel()	12
4.4 ElectricalLoad Class Reference	12
4.4.1 Detailed Description	12
4.4.2 Constructor & Destructor Documentation	12
4.4.2.1 ElectricalLoad()	12
4.4.2.2 ~ElectricalLoad()	13
4.5 Lilon Class Reference	13
4.5.1 Detailed Description	14
4.5.2 Constructor & Destructor Documentation	14
4.5.2.1 Lilon()	14
4.5.2.2 ~Lilon()	14
4.6 Model Class Reference	15
4.6.1 Detailed Description	15
4.6.2 Constructor & Destructor Documentation	16
4.6.2.1 Model()	16
4.6.2.2 ~Model()	16
4.6.3 Member Data Documentation	16
4.6.3.1 combustion_ptr_vec	16
4.6.3.2 controller	16

4.6.3.3 electrical_load	17
4.6.3.4 renewable_ptr_vec	17
4.6.3.5 resources	17
4.6.3.6 storage_ptr_vec	17
4.7 Production Class Reference	17
4.7.1 Detailed Description	18
4.7.2 Constructor & Destructor Documentation	18
4.7.2.1 Production()	18
4.7.2.2 ~Production()	18
4.8 Renewable Class Reference	19
4.8.1 Detailed Description	20
4.8.2 Constructor & Destructor Documentation	20
4.8.2.1 Renewable()	20
4.8.2.2 ~Renewable()	20
4.9 Resources Class Reference	20
4.9.1 Detailed Description	21
4.9.2 Constructor & Destructor Documentation	21
4.9.2.1 Resources()	21
4.9.2.2 ~Resources()	21
4.10 Solar Class Reference	21
4.10.1 Detailed Description	22
4.10.2 Constructor & Destructor Documentation	23
4.10.2.1 Solar()	23
4.10.2.2 ~Solar()	23
4.11 Storage Class Reference	23
4.11.1 Detailed Description	24
4.11.2 Constructor & Destructor Documentation	24
4.11.2.1 Storage()	24
4.11.2.2 ~Storage()	24
4.12 Tidal Class Reference	25
4.12.1 Detailed Description	26
4.12.2 Constructor & Destructor Documentation	26
4.12.2.1 Tidal()	26
4.12.2.2 ~Tidal()	26
4.13 Wave Class Reference	27
4.13.1 Detailed Description	28
4.13.2 Constructor & Destructor Documentation	28
4.13.2.1 Wave()	28
4.13.2.2 ~Wave()	28
4.14 Wind Class Reference	29
4.14.1 Detailed Description	30
4.14.2 Constructor & Destructor Documentation	30

4.14.2.1 Wind()	30
4.14.2.2 ~Wind()	30
<b>5 File Documentation</b>	<b>31</b>
5.1 header/Controller.h File Reference	31
5.1.1 Detailed Description	32
5.2 header/ElectricalLoad.h File Reference	32
5.2.1 Detailed Description	33
5.3 header/Model.h File Reference	33
5.3.1 Detailed Description	34
5.4 header/Production/Combustion/Combustion.h File Reference	34
5.4.1 Detailed Description	34
5.5 header/Production/Combustion/Diesel.h File Reference	35
5.5.1 Detailed Description	35
5.6 header/Production/Production.h File Reference	36
5.6.1 Detailed Description	36
5.7 header/Production/Renewable/Renewable.h File Reference	36
5.7.1 Detailed Description	37
5.8 header/Production/Renewable/Solar.h File Reference	37
5.8.1 Detailed Description	38
5.9 header/Production/Renewable/Tidal.h File Reference	38
5.9.1 Detailed Description	39
5.10 header/Production/Renewable/Wave.h File Reference	39
5.10.1 Detailed Description	40
5.11 header/Production/Renewable/Wind.h File Reference	40
5.11.1 Detailed Description	41
5.12 header/Resources.h File Reference	41
5.12.1 Detailed Description	42
5.13 header/std_includes.h File Reference	42
5.13.1 Detailed Description	43
5.14 header/Storage/Lilon.h File Reference	43
5.14.1 Detailed Description	44
5.15 header/Storage/Storage.h File Reference	44
5.15.1 Detailed Description	45
5.16 pybindings/PYBIND11_PGM.cpp File Reference	45
5.16.1 Detailed Description	45
5.16.2 Function Documentation	45
5.16.2.1 PYBIND11_MODULE()	46
5.17 source/Controller.cpp File Reference	46
5.17.1 Detailed Description	47
5.18 source/ElectricalLoad.cpp File Reference	47
5.18.1 Detailed Description	47

5.19 source/Model.cpp File Reference	47
5.19.1 Detailed Description	48
5.20 source/Production/Combustion/Combustion.cpp File Reference	48
5.20.1 Detailed Description	48
5.21 source/Production/Combustion/Diesel.cpp File Reference	48
5.21.1 Detailed Description	49
5.22 source/Production/Production.cpp File Reference	49
5.22.1 Detailed Description	49
5.23 source/Production/Renewable/Renewable.cpp File Reference	49
5.23.1 Detailed Description	50
5.24 source/Production/Renewable/Solar.cpp File Reference	50
5.24.1 Detailed Description	50
5.25 source/Production/Renewable/Tidal.cpp File Reference	50
5.25.1 Detailed Description	51
5.26 source/Production/Renewable/Wave.cpp File Reference	51
5.26.1 Detailed Description	51
5.27 source/Production/Renewable/Wind.cpp File Reference	51
5.27.1 Detailed Description	52
5.28 source/Resources.cpp File Reference	52
5.28.1 Detailed Description	52
5.29 source/Storage/Lilon.cpp File Reference	52
5.29.1 Detailed Description	53
5.30 source/Storage/Storage.cpp File Reference	53
5.30.1 Detailed Description	53
5.31 test/utlis/testing_utils.cpp File Reference	53
5.31.1 Detailed Description	54
5.31.2 Function Documentation	54
5.31.2.1 expectedErrorNotDetected()	54
5.31.2.2 printGold()	55
5.31.2.3 printGreen()	55
5.31.2.4 printRed()	55
5.31.2.5 testFloatEquals()	56
5.31.2.6 testGreaterThan()	56
5.31.2.7 testGreaterThanOrEqualTo()	57
5.31.2.8 testLessThan()	58
5.31.2.9 testLessThanOrEqualTo()	58
5.31.2.10 testTruth()	59
5.32 test/utlis/testing_utils.h File Reference	59
5.32.1 Detailed Description	61
5.32.2 Macro Definition Documentation	61
5.32.2.1 FLOAT_TOLERANCE	61
5.32.3 Function Documentation	61

---

5.32.3.1 expectedErrorNotDetected()	61
5.32.3.2 printGold()	62
5.32.3.3 printGreen()	62
5.32.3.4 printRed()	62
5.32.3.5 testFloatEquals()	63
5.32.3.6 testGreaterThan()	63
5.32.3.7 testGreaterThanOrEqualTo()	64
5.32.3.8 testLessThan()	65
5.32.3.9 testLessThanOrEqualTo()	65
5.32.3.10 testTruth()	66

<b>Index</b>	<b>67</b>
--------------	-----------





# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Controller . . . . .	9
ElectricalLoad . . . . .	12
Model . . . . .	15
Production . . . . .	17
Combustion . . . . .	7
Diesel . . . . .	10
Renewable . . . . .	19
Solar . . . . .	21
Tidal . . . . .	25
Wave . . . . .	27
Wind . . . . .	29
Resources . . . . .	20
Storage . . . . .	23
Lilon . . . . .	13



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Combustion</a>	The root of the <a href="#">Combustion</a> branch of the <a href="#">Production</a> hierarchy. This branch contains derived classes which model the production of energy by way of combustibles . . . . .	7
<a href="#">Controller</a>	A class which contains a various dispatch control logic. Intended to serve as a component class of <a href="#">Model</a> . . . . .	9
<a href="#">Diesel</a>	A derived class of the <a href="#">Combustion</a> branch of <a href="#">Production</a> which models production using a diesel generator . . . . .	10
<a href="#">ElectricalLoad</a>	A class which contains time and electrical load data. Intended to serve as a component class of <a href="#">Model</a> . . . . .	12
<a href="#">Lilon</a>	A derived class of <a href="#">Storage</a> which models energy storage by way of lithium-ion batteries . . . .	13
<a href="#">Model</a>	A container class which forms the centre of PGMcpp. The <a href="#">Model</a> class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes . . . . .	15
<a href="#">Production</a>	The base class of the <a href="#">Production</a> hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise . . . . .	17
<a href="#">Renewable</a>	The root of the <a href="#">Renewable</a> branch of the <a href="#">Production</a> hierarchy. This branch contains derived classes which model the renewable production of energy . . . . .	19
<a href="#">Resources</a>	A class which contains renewable resource data. Intended to serve as a component class of <a href="#">Model</a> . . . . .	20
<a href="#">Solar</a>	A derived class of the <a href="#">Renewable</a> branch of <a href="#">Production</a> which models solar production . . . .	21
<a href="#">Storage</a>	The base class of the <a href="#">Storage</a> hierarchy. This hierarchy contains derived classes which model the storage of energy . . . . .	23
<a href="#">Tidal</a>	A derived class of the <a href="#">Renewable</a> branch of <a href="#">Production</a> which models tidal production . . . .	25
<a href="#">Wave</a>	A derived class of the <a href="#">Renewable</a> branch of <a href="#">Production</a> which models wave production . . . .	27
<a href="#">Wind</a>	A derived class of the <a href="#">Renewable</a> branch of <a href="#">Production</a> which models wind production . . . .	29



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

header/ <a href="#">Controller.h</a>	
Header file the <a href="#">Controller</a> class . . . . .	31
header/ <a href="#">ElectricalLoad.h</a>	
Header file the <a href="#">ElectricalLoad</a> class . . . . .	32
header/ <a href="#">Model.h</a>	
Header file the <a href="#">Model</a> class . . . . .	33
header/ <a href="#">Resources.h</a>	
Header file the <a href="#">Resources</a> class . . . . .	41
header/ <a href="#">std_includes.h</a>	
Header file which simply batches together the usual, standard includes . . . . .	42
header/Production/ <a href="#">Production.h</a>	
Header file the <a href="#">Production</a> class . . . . .	36
header/Production/Combustion/ <a href="#">Combustion.h</a>	
Header file the <a href="#">Combustion</a> class . . . . .	34
header/Production/Combustion/ <a href="#">Diesel.h</a>	
Header file the <a href="#">Diesel</a> class . . . . .	35
header/Production/Renewable/ <a href="#">Renewable.h</a>	
Header file the <a href="#">Renewable</a> class . . . . .	36
header/Production/Renewable/ <a href="#">Solar.h</a>	
Header file the <a href="#">Solar</a> class . . . . .	37
header/Production/Renewable/ <a href="#">Tidal.h</a>	
Header file the <a href="#">Tidal</a> class . . . . .	38
header/Production/Renewable/ <a href="#">Wave.h</a>	
Header file the <a href="#">Wave</a> class . . . . .	39
header/Production/Renewable/ <a href="#">Wind.h</a>	
Header file the <a href="#">Wind</a> class . . . . .	40
header/Storage/ <a href="#">Lilon.h</a>	
Header file the <a href="#">Lilon</a> class . . . . .	43
header/Storage/ <a href="#">Storage.h</a>	
Header file the <a href="#">Storage</a> class . . . . .	44
pybindings/ <a href="#">PYBIND11_PGM.cpp</a>	
Python 3 bindings file for PGMcpp . . . . .	45
source/ <a href="#">Controller.cpp</a>	
Implementation file for the <a href="#">Controller</a> class . . . . .	46
source/ <a href="#">ElectricalLoad.cpp</a>	
Implementation file for the <a href="#">ElectricalLoad</a> class . . . . .	47

source/ <a href="#">Model.cpp</a>	
Implementation file for the <a href="#">Model</a> class . . . . .	47
source/ <a href="#">Resources.cpp</a>	
Implementation file for the <a href="#">Resources</a> class . . . . .	52
source/Production/ <a href="#">Production.cpp</a>	
Implementation file for the <a href="#">Production</a> class . . . . .	49
source/Production/Combustion/ <a href="#">Combustion.cpp</a>	
Implementation file for the <a href="#">Combustion</a> class . . . . .	48
source/Production/Combustion/ <a href="#">Diesel.cpp</a>	
Implementation file for the <a href="#">Diesel</a> class . . . . .	48
source/Production/Renewable/ <a href="#">Renewable.cpp</a>	
Implementation file for the <a href="#">Renewable</a> class . . . . .	49
source/Production/Renewable/ <a href="#">Solar.cpp</a>	
Implementation file for the <a href="#">Solar</a> class . . . . .	50
source/Production/Renewable/ <a href="#">Tidal.cpp</a>	
Implementation file for the <a href="#">Tidal</a> class . . . . .	50
source/Production/Renewable/ <a href="#">Wave.cpp</a>	
Implementation file for the <a href="#">Wave</a> class . . . . .	51
source/Production/Renewable/ <a href="#">Wind.cpp</a>	
Implementation file for the <a href="#">Wind</a> class . . . . .	51
source/Storage/ <a href="#">Lilon.cpp</a>	
Implementation file for the <a href="#">Lilon</a> class . . . . .	52
source/Storage/ <a href="#">Storage.cpp</a>	
Implementation file for the <a href="#">Storage</a> class . . . . .	53
test/utills/ <a href="#">testing_utils.cpp</a>	
Header file for various PGMcpp testing utilities . . . . .	53
test/utills/ <a href="#">testing_utils.h</a>	
Header file for various PGMcpp testing utilities . . . . .	59

## Chapter 4

# Class Documentation

### 4.1 Combustion Class Reference

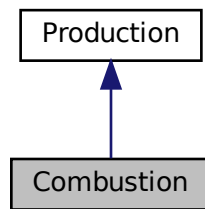
The root of the [Combustion](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

```
#include <Combustion.h>
```

Inheritance diagram for Combustion:



Collaboration diagram for Combustion:



## Public Member Functions

- [Combustion](#) (void)  
*Constructor for the [Combustion](#) class.*
- virtual [~Combustion](#) (void)  
*Destructor for the [Combustion](#) class.*

### 4.1.1 Detailed Description

The root of the [Combustion](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Combustion()

```
Combustion::Combustion (  
    void )
```

Constructor for the [Combustion](#) class.

```
36      :  
37 Production()  
38 {  
39     //...  
40  
41     return;  
42 } /* Combustion() */
```



#### 4.1.2.2 ~Combustion()

```
Combustion::~Combustion (
    void ) [virtual]
```

Destructor for the [Combustion](#) class.

```
65 {
66     //...
67
68     return;
69 } /* ~Combustion() */
```

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

- header/Production/Combustion/[Combustion.h](#)
- source/Production/Combustion/[Combustion.cpp](#)

## 4.2 Controller Class Reference

A class which contains a various dispatch control logic. Intended to serve as a component class of [Model](#).

```
#include <Controller.h>
```

### Public Member Functions

- [Controller](#) (void)  
*Constructor for the [Controller](#) class.*
- [~Controller](#) (void)  
*Destructor for the [Controller](#) class.*

#### 4.2.1 Detailed Description

A class which contains a various dispatch control logic. Intended to serve as a component class of [Model](#).

#### 4.2.2 Constructor & Destructor Documentation

##### 4.2.2.1 Controller()

```
Controller::Controller (
    void )
```

Constructor for the [Controller](#) class.

```
36 {
37     //...
38
39     return;
40 } /* Controller() */
```

#### 4.2.2.2 ~Controller()

```
Controller::~~Controller (
    void )
```

Destructor for the [Controller](#) class.

```
63 {
64     //...
65
66     return;
67 } /* ~Controller() */
```

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

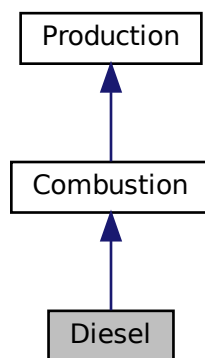
- header/[Controller.h](#)
- source/[Controller.cpp](#)

## 4.3 Diesel Class Reference

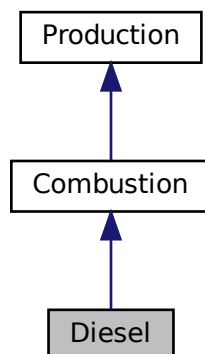
A derived class of the [Combustion](#) branch of [Production](#) which models production using a diesel generator.

```
#include <Diesel.h>
```

Inheritance diagram for Diesel:



Collaboration diagram for Diesel:



## Public Member Functions

- [Diesel](#) (void)  
*Constructor for the [Diesel](#) class.*
- [~Diesel](#) (void)  
*Destructor for the [Diesel](#) class.*

### 4.3.1 Detailed Description

A derived class of the [Combustion](#) branch of [Production](#) which models production using a diesel generator.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Diesel()

```
Diesel::Diesel (  
    void )
```

Constructor for the [Diesel](#) class.

```
35         :  
36     Combustion()  
37     {  
38         //...  
39     }  
40     return;  
41 } /* Diesel() */
```

#### 4.3.2.2 ~Diesel()

```
Diesel::~~Diesel (
    void )
```

Destructor for the [Diesel](#) class.

```
64 {
65     //...
66
67     return;
68 } /* ~Diesel() */
```

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

- header/Production/Combustion/[Diesel.h](#)
- source/Production/Combustion/[Diesel.cpp](#)

## 4.4 ElectricalLoad Class Reference

A class which contains time and electrical load data. Intended to serve as a component class of [Model](#).

```
#include <ElectricalLoad.h>
```

### Public Member Functions

- [ElectricalLoad](#) (void)  
*Constructor for the [ElectricalLoad](#) class.*
- [~ElectricalLoad](#) (void)  
*Destructor for the [ElectricalLoad](#) class.*

#### 4.4.1 Detailed Description

A class which contains time and electrical load data. Intended to serve as a component class of [Model](#).

#### 4.4.2 Constructor & Destructor Documentation

##### 4.4.2.1 ElectricalLoad()

```
ElectricalLoad::ElectricalLoad (
    void )
```

Constructor for the [ElectricalLoad](#) class.

```
36 {
37     //...
38
39     return;
40 } /* ElectricalLoad() */
```

#### 4.4.2.2 ~ElectricalLoad()

```
ElectricalLoad::~ElectricalLoad (
    void )
```

Destructor for the [ElectricalLoad](#) class.

```
63 {
64     //...
65
66     return;
67 } /* ~ElectricalLoad() */
```

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

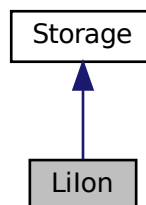
- header/[ElectricalLoad.h](#)
- source/[ElectricalLoad.cpp](#)

## 4.5 Lilon Class Reference

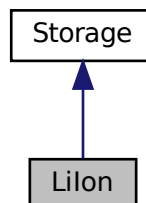
A derived class of [Storage](#) which models energy storage by way of lithium-ion batteries.

```
#include <LiIon.h>
```

Inheritance diagram for Lilon:



Collaboration diagram for Lilon:



## Public Member Functions

- [Lilon](#) (void)  
*Constructor for the [Lilon](#) class.*
- [~Lilon](#) (void)  
*Destructor for the [Lilon](#) class.*

### 4.5.1 Detailed Description

A derived class of [Storage](#) which models energy storage by way of lithium-ion batteries.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 Lilon()

```
LiIon::LiIon (  
    void )
```

Constructor for the [Lilon](#) class.

```
35      :  
36  Storage()  
37  {  
38      //...  
39  
40      return;  
41  } /* LiIon() */
```

#### 4.5.2.2 ~Lilon()

```
LiIon::~LiIon (  
    void )
```

Destructor for the [Lilon](#) class.

```
64 {  
65     //...  
66  
67     return;  
68 } /* ~LiIon() */
```

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

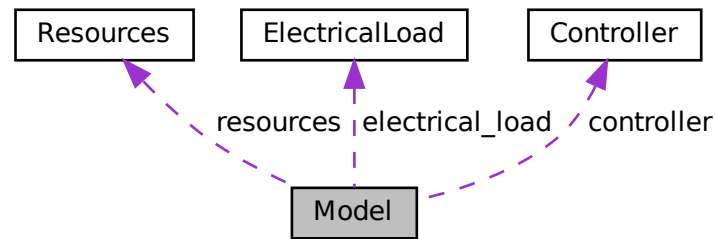
- header/Storage/[Lilon.h](#)
- source/Storage/[Lilon.cpp](#)

## 4.6 Model Class Reference

A container class which forms the centre of PGMcpp. The [Model](#) class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes.

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

Collaboration diagram for Model:



### Public Member Functions

- [Model](#) (void)  
*Constructor for the [Model](#) class.*
- [~Model](#) (void)  
*Destructor for the [Model](#) class.*

### Public Attributes

- [Controller](#) controller  
*[Controller](#) component of [Model](#).*
- [ElectricalLoad](#) electrical\_load  
*[ElectricalLoad](#) component of [Model](#).*
- [Resources](#) resources  
*[Resources](#) component of [Model](#).*
- `std::vector< Combustion * > combustion_ptr_vec`  
*A vector of pointers to the various [Combustion](#) assets in the [Model](#).*
- `std::vector< Renewable * > renewable_ptr_vec`  
*A vector of pointers to the various [Renewable](#) assets in the [Model](#).*
- `std::vector< Storage * > storage_ptr_vec`  
*A vector of pointers to the various [Storage](#) assets in the [Model](#).*

### 4.6.1 Detailed Description

A container class which forms the centre of PGMcpp. The [Model](#) class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes.

## 4.6.2 Constructor & Destructor Documentation

### 4.6.2.1 Model()

```
Model::Model (  
    void )
```

Constructor for the [Model](#) class.

```
37 {  
38     //...  
39  
40     return;  
41 } /* Model() */
```

### 4.6.2.2 ~Model()

```
Model::~~Model (  
    void )
```

Destructor for the [Model](#) class.

```
64 {  
65     //...  
66  
67     return;  
68 } /* ~Model() */
```

## 4.6.3 Member Data Documentation

### 4.6.3.1 combustion\_ptr\_vec

```
std::vector<Combustion*> Model::combustion_ptr_vec
```

A vector of pointers to the various [Combustion](#) assets in the [Model](#).

### 4.6.3.2 controller

```
Controller Model::controller
```

[Controller](#) component of [Model](#).



#### 4.6.3.3 electrical\_load

`ElectricalLoad` `Model::electrical_load`

`ElectricalLoad` component of `Model`.

#### 4.6.3.4 renewable\_ptr\_vec

`std::vector<Renewable*>` `Model::renewable_ptr_vec`

A vector of pointers to the various `Renewable` assets in the `Model`.

#### 4.6.3.5 resources

`Resources` `Model::resources`

`Resources` component of `Model`.

#### 4.6.3.6 storage\_ptr\_vec

`std::vector<Storage*>` `Model::storage_ptr_vec`

A vector of pointers to the various `Storage` assets in the `Model`.

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

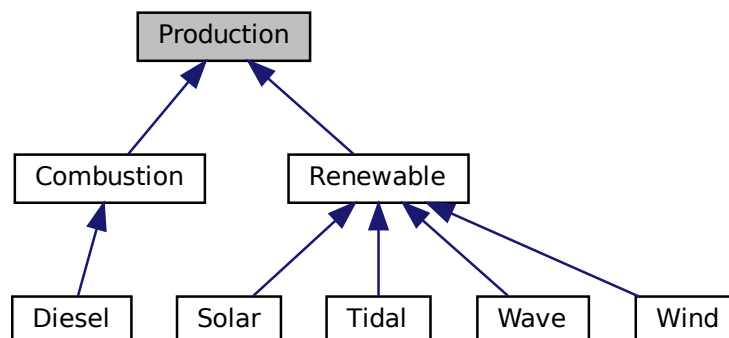
- header/`Model.h`
- source/`Model.cpp`

## 4.7 Production Class Reference

The base class of the `Production` hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise.

```
#include <Production.h>
```

Inheritance diagram for `Production`:



## Public Member Functions

- [Production](#) (void)  
*Constructor for the [Production](#) class.*
- virtual [~Production](#) (void)  
*Destructor for the [Production](#) class.*

### 4.7.1 Detailed Description

The base class of the [Production](#) hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 Production()

```
Production::Production (  
    void )
```

Constructor for the [Production](#) class.

```
36 {  
37     //...  
38  
39     return;  
40 } /* Production() */
```

#### 4.7.2.2 ~Production()

```
Production::~~Production (  
    void ) [virtual]
```

Destructor for the [Production](#) class.

```
63 {  
64     //...  
65  
66     return;  
67 } /* ~Production() */
```

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

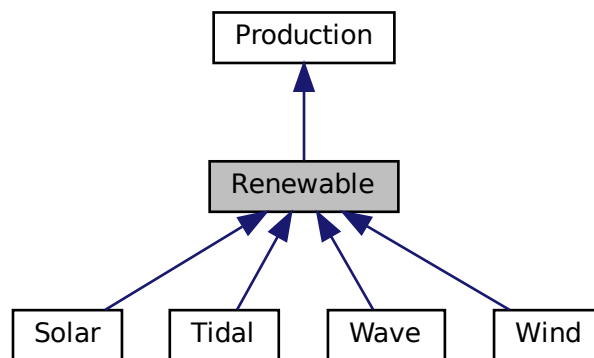
- header/Production/[Production.h](#)
- source/Production/[Production.cpp](#)

## 4.8 Renewable Class Reference

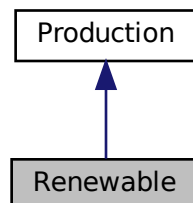
The root of the [Renewable](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the renewable production of energy.

```
#include <Renewable.h>
```

Inheritance diagram for Renewable:



Collaboration diagram for Renewable:



### Public Member Functions

- [Renewable](#) (void)  
*Constructor for the [Renewable](#) class.*
- virtual [~Renewable](#) (void)  
*Destructor for the [Renewable](#) class.*

### 4.8.1 Detailed Description

The root of the [Renewable](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the renewable production of energy.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 Renewable()

```
Renewable::Renewable (  
    void )
```

Constructor for the [Renewable](#) class.

```
35      :  
36  Production()  
37  {  
38      //...  
39  
40      return;  
41  } /* Renewable() */
```

#### 4.8.2.2 ~Renewable()

```
Renewable::~~Renewable (  
    void ) [virtual]
```

Destructor for the [Renewable](#) class.

```
64 {  
65     //...  
66  
67     return;  
68 } /* ~Renewable() */
```

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

- header/Production/Renewable/[Renewable.h](#)
- source/Production/Renewable/[Renewable.cpp](#)

## 4.9 Resources Class Reference

A class which contains renewable resource data. Intended to serve as a component class of [Model](#).

```
#include <Resources.h>
```

### Public Member Functions

- [Resources](#) (void)  
*Constructor for the [Resources](#) class.*
- [~Resources](#) (void)  
*Destructor for the [Resources](#) class.*

### 4.9.1 Detailed Description

A class which contains renewable resource data. Intended to serve as a component class of [Model](#).

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 Resources()

```
Resources::Resources (  
    void )
```

Constructor for the [Resources](#) class.

```
36 {  
37     //...  
38  
39     return;  
40 } /* Resources() */
```

#### 4.9.2.2 ~Resources()

```
Resources::~~Resources (  
    void )
```

Destructor for the [Resources](#) class.

```
63 {  
64     //...  
65  
66     return;  
67 } /* ~Resources() */
```

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

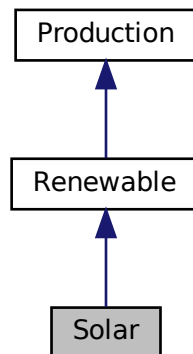
- header/[Resources.h](#)
- source/[Resources.cpp](#)

## 4.10 Solar Class Reference

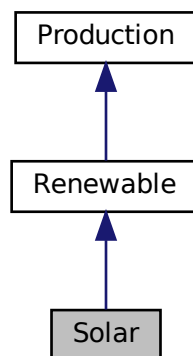
A derived class of the [Renewable](#) branch of [Production](#) which models solar production.

```
#include <Solar.h>
```

Inheritance diagram for Solar:



Collaboration diagram for Solar:



## Public Member Functions

- [Solar](#) (void)  
*Constructor for the [Solar](#) class.*
- [~Solar](#) (void)  
*Destructor for the [Solar](#) class.*

### 4.10.1 Detailed Description

A derived class of the [Renewable](#) branch of [Production](#) which models solar production.

## 4.10.2 Constructor & Destructor Documentation

### 4.10.2.1 Solar()

```
Solar::Solar (
    void )
```

Constructor for the [Solar](#) class.

```
35     :
36     Renewable()
37 {
38     //...
39
40     return;
41 } /* Solar() */
```

### 4.10.2.2 ~Solar()

```
Solar::~~Solar (
    void )
```

Destructor for the [Solar](#) class.

```
64 {
65     //...
66
67     return;
68 } /* ~Solar() */
```

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

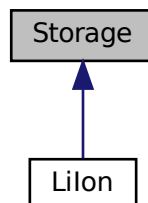
- [header/Production/Renewable/Solar.h](#)
- [source/Production/Renewable/Solar.cpp](#)

## 4.11 Storage Class Reference

The base class of the [Storage](#) hierarchy. This hierarchy contains derived classes which model the storage of energy.

```
#include <Storage.h>
```

Inheritance diagram for Storage:



## Public Member Functions

- [Storage](#) (void)  
*Constructor for the [Storage](#) class.*
- virtual [~Storage](#) (void)  
*Destructor for the [Storage](#) class.*

### 4.11.1 Detailed Description

The base class of the [Storage](#) hierarchy. This hierarchy contains derived classes which model the storage of energy.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 Storage()

```
Storage::Storage (  
    void )
```

Constructor for the [Storage](#) class.

```
36 {  
37     //...  
38  
39     return;  
40 } /* Storage() */
```

#### 4.11.2.2 ~Storage()

```
Storage::~~Storage (  
    void ) [virtual]
```

Destructor for the [Storage](#) class.

```
63 {  
64     //...  
65  
66     return;  
67 } /* ~Storage() */
```

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

- header/Storage/[Storage.h](#)
- source/Storage/[Storage.cpp](#)

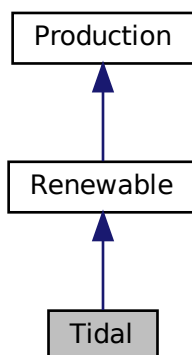


## 4.12 Tidal Class Reference

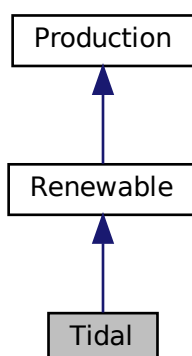
A derived class of the [Renewable](#) branch of [Production](#) which models tidal production.

```
#include <Tidal.h>
```

Inheritance diagram for Tidal:



Collaboration diagram for Tidal:



### Public Member Functions

- [Tidal](#) (void)  
*Constructor for the [Tidal](#) class.*
- [~Tidal](#) (void)  
*Destructor for the [Tidal](#) class.*

### 4.12.1 Detailed Description

A derived class of the [Renewable](#) branch of [Production](#) which models tidal production.

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 Tidal()

```
Tidal::Tidal (  
    void )
```

Constructor for the [Tidal](#) class.

```
35     :  
36     Renewable()  
37 {  
38     //...  
39  
40     return;  
41 } /* Tidal() */
```

#### 4.12.2.2 ~Tidal()

```
Tidal::~~Tidal (  
    void )
```

Destructor for the [Tidal](#) class.

```
64 {  
65     //...  
66  
67     return;  
68 } /* ~Tidal() */
```

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

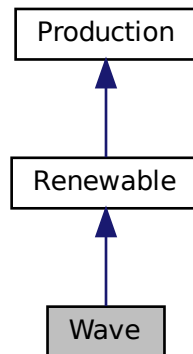
- [header/Production/Renewable/Tidal.h](#)
- [source/Production/Renewable/Tidal.cpp](#)

## 4.13 Wave Class Reference

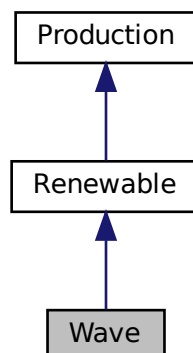
A derived class of the [Renewable](#) branch of [Production](#) which models wave production.

```
#include <Wave.h>
```

Inheritance diagram for Wave:



Collaboration diagram for Wave:



### Public Member Functions

- [Wave](#) (void)  
*Constructor for the [Wave](#) class.*
- [~Wave](#) (void)  
*Destructor for the [Wave](#) class.*

### 4.13.1 Detailed Description

A derived class of the [Renewable](#) branch of [Production](#) which models wave production.

### 4.13.2 Constructor & Destructor Documentation

#### 4.13.2.1 Wave()

```
Wave::Wave (
    void )
```

Constructor for the [Wave](#) class.

```
35     :
36     Renewable()
37 {
38     //...
39
40     return;
41 } /* Wave() */
```

#### 4.13.2.2 ~Wave()

```
Wave::~~Wave (
    void )
```

Destructor for the [Wave](#) class.

```
64 {
65     //...
66
67     return;
68 } /* ~Wave() */
```

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

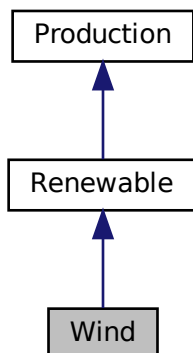
- [header/Production/Renewable/Wave.h](#)
- [source/Production/Renewable/Wave.cpp](#)

## 4.14 Wind Class Reference

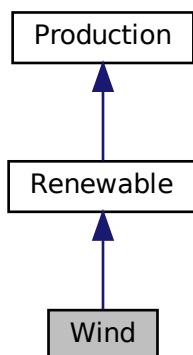
A derived class of the [Renewable](#) branch of [Production](#) which models wind production.

```
#include <Wind.h>
```

Inheritance diagram for Wind:



Collaboration diagram for Wind:



### Public Member Functions

- [Wind](#) (void)  
*Constructor for the [Wind](#) class.*
- [~Wind](#) (void)  
*Destructor for the [Wind](#) class.*

### 4.14.1 Detailed Description

A derived class of the [Renewable](#) branch of [Production](#) which models wind production.

### 4.14.2 Constructor & Destructor Documentation

#### 4.14.2.1 Wind()

```
Wind::Wind (
    void )
```

Constructor for the [Wind](#) class.

```
35     :
36     Renewable()
37 {
38     //...
39
40     return;
41 } /* Wind() */
```

#### 4.14.2.2 ~Wind()

```
Wind::~~Wind (
    void )
```

Destructor for the [Wind](#) class.

```
64 {
65     //...
66
67     return;
68 } /* ~Wind() */
```

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

- [header/Production/Renewable/Wind.h](#)
- [source/Production/Renewable/Wind.cpp](#)

## Chapter 5

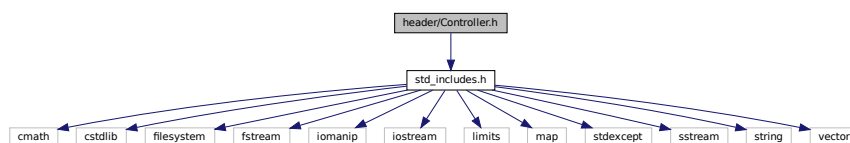
# File Documentation

### 5.1 header/Controller.h File Reference

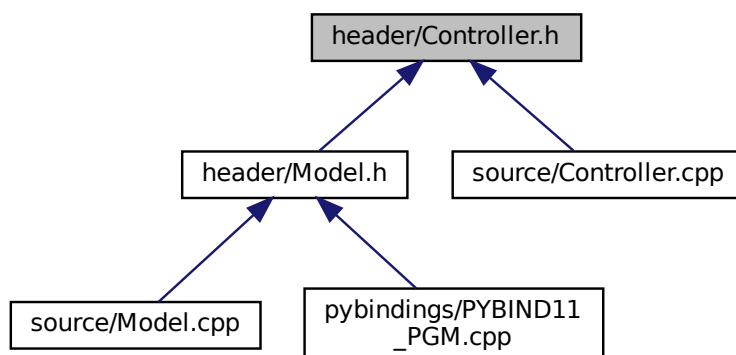
Header file the [Controller](#) class.

```
#include "std_includes.h"
```

Include dependency graph for Controller.h:



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



## Classes

- class [Controller](#)

A class which contains a various dispatch control logic. Intended to serve as a component class of [Model](#).

### 5.1.1 Detailed Description

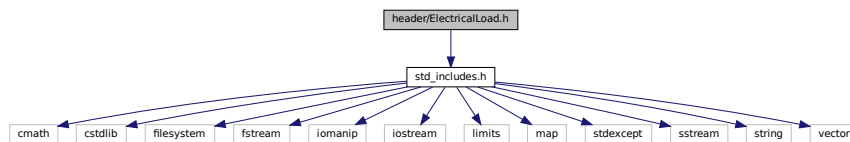
Header file the [Controller](#) class.

## 5.2 header/ElectricalLoad.h File Reference

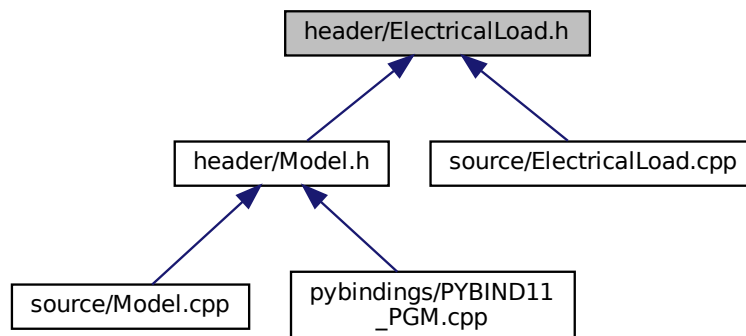
Header file the [ElectricalLoad](#) class.

```
#include "std_includes.h"
```

Include dependency graph for ElectricalLoad.h:



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



## Classes

- class [ElectricalLoad](#)

A class which contains time and electrical load data. Intended to serve as a component class of [Model](#).



### 5.2.1 Detailed Description

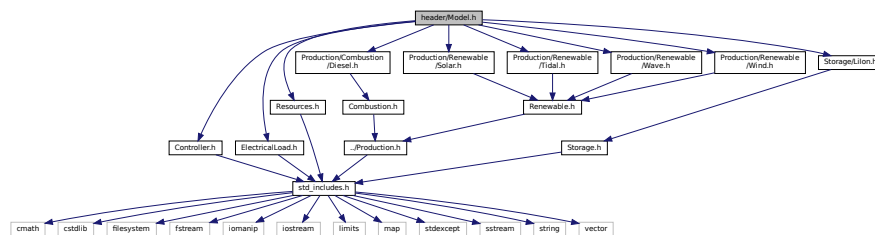
Header file the [ElectricalLoad](#) class.

## 5.3 header/Model.h File Reference

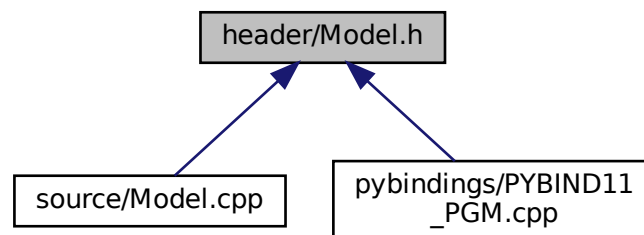
Header file the [Model](#) class.

```
#include "Controller.h"
#include "ElectricalLoad.h"
#include "Resources.h"
#include "Production/Combustion/Diesel.h"
#include "Production/Renewable/Solar.h"
#include "Production/Renewable/Tidal.h"
#include "Production/Renewable/Wave.h"
#include "Production/Renewable/Wind.h"
#include "Storage/LiIon.h"
```

Include dependency graph for Model.h:



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



## Classes

- class [Model](#)

*A container class which forms the centre of PGMcpp. The [Model](#) class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes.*

### 5.3.1 Detailed Description

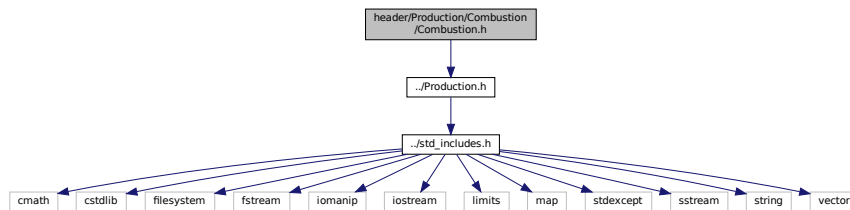
Header file the [Model](#) class.

## 5.4 header/Production/Combustion/Combustion.h File Reference

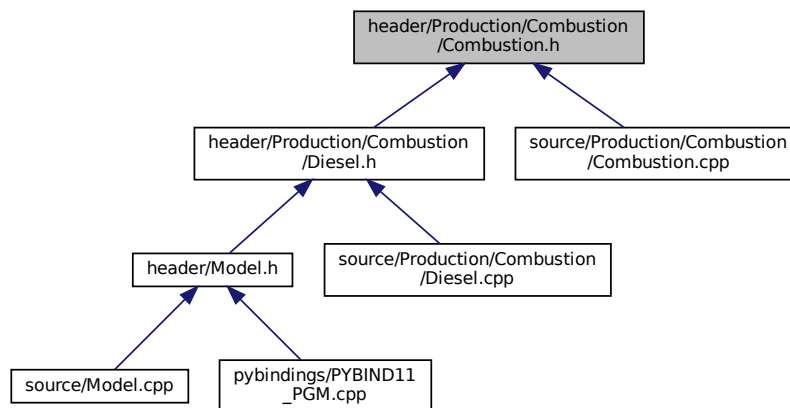
Header file the [Combustion](#) class.

```
#include "../Production.h"
```

Include dependency graph for Combustion.h:



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



## Classes

- class [Combustion](#)

*The root of the [Combustion](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.*

### 5.4.1 Detailed Description

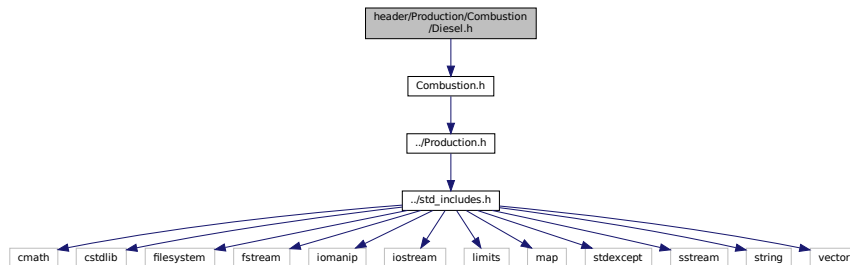
Header file the [Combustion](#) class.

## 5.5 header/Production/Combustion/Diesel.h File Reference

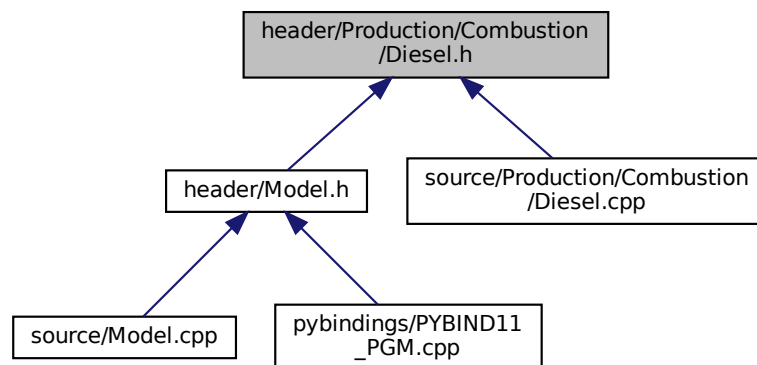
Header file the [Diesel](#) class.

```
#include "Combustion.h"
```

Include dependency graph for Diesel.h:



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



### Classes

- class [Diesel](#)

*A derived class of the [Combustion](#) branch of [Production](#) which models production using a diesel generator.*

### 5.5.1 Detailed Description

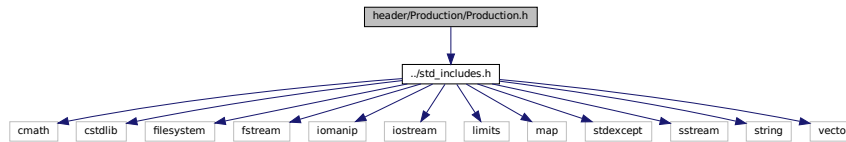
Header file the [Diesel](#) class.

## 5.6 header/Production/Production.h File Reference

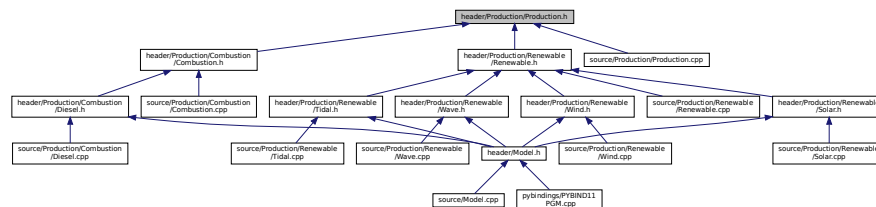
Header file the [Production](#) class.

```
#include "../std_includes.h"
```

Include dependency graph for Production.h:



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



### Classes

- class [Production](#)

*The base class of the [Production](#) hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise.*

### 5.6.1 Detailed Description

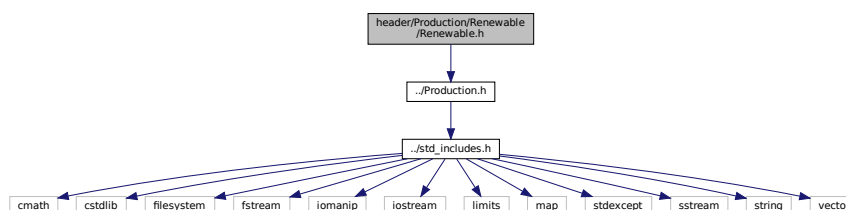
Header file the [Production](#) class.

## 5.7 header/Production/Renewable/Renewable.h File Reference

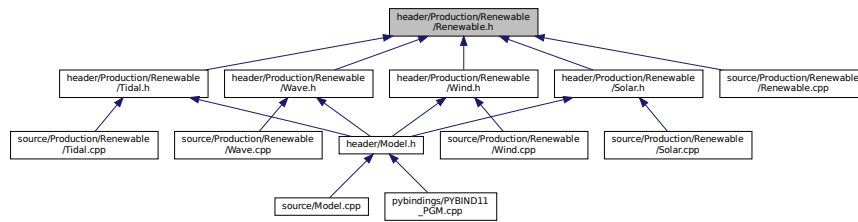
Header file the [Renewable](#) class.

```
#include "../Production.h"
```

Include dependency graph for Renewable.h:



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



## Classes

- class [Renewable](#)

The root of the [Renewable](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the renewable production of energy.

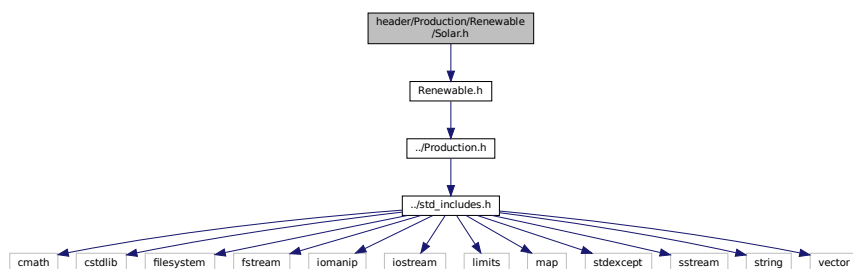
### 5.7.1 Detailed Description

Header file the [Renewable](#) class.

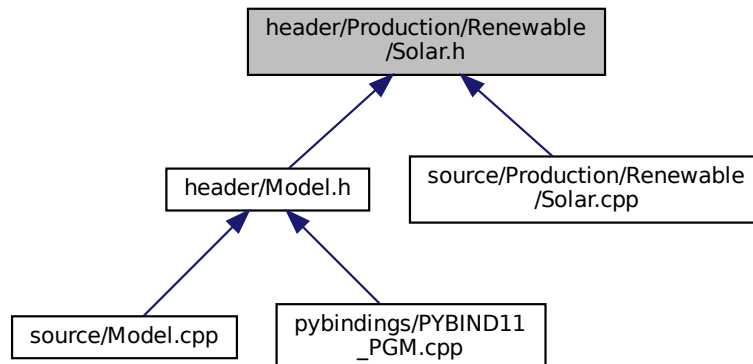
## 5.8 header/Production/Renewable/Solar.h File Reference

Header file the [Solar](#) class.

```
#include "Renewable.h"
Include dependency graph for Solar.h:
```



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



## Classes

- class [Solar](#)

*A derived class of the [Renewable](#) branch of [Production](#) which models solar production.*

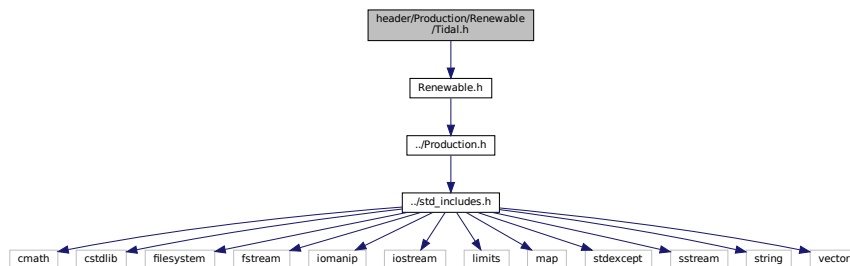
### 5.8.1 Detailed Description

Header file the [Solar](#) class.

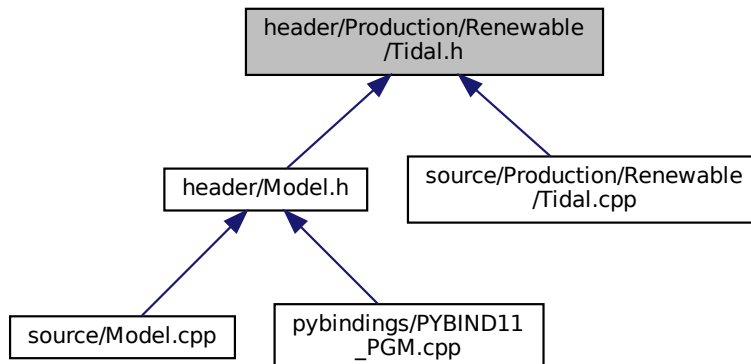
## 5.9 header/Production/Renewable/Tidal.h File Reference

Header file the [Tidal](#) class.

```
#include "Renewable.h"
Include dependency graph for Tidal.h:
```



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



## Classes

- class [Tidal](#)

A derived class of the [Renewable](#) branch of [Production](#) which models tidal production.

### 5.9.1 Detailed Description

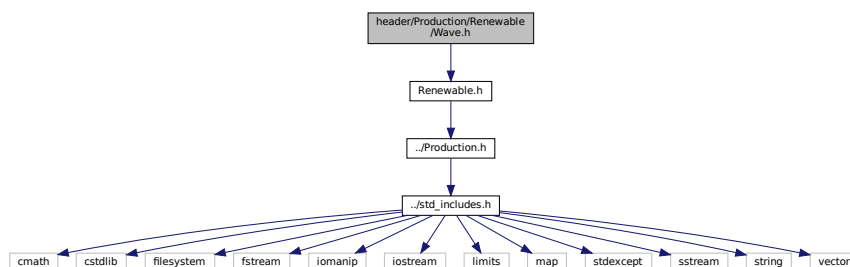
Header file the [Tidal](#) class.

## 5.10 header/Production/Renewable/Wave.h File Reference

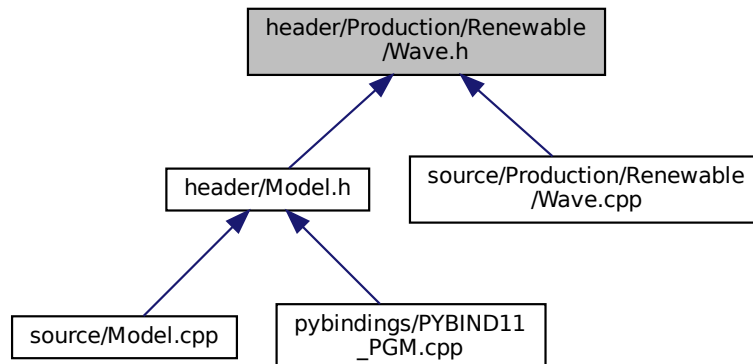
Header file the [Wave](#) class.

```
#include "Renewable.h"
```

Include dependency graph for Wave.h:



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



## Classes

- class [Wave](#)

*A derived class of the [Renewable](#) branch of [Production](#) which models wave production.*

### 5.10.1 Detailed Description

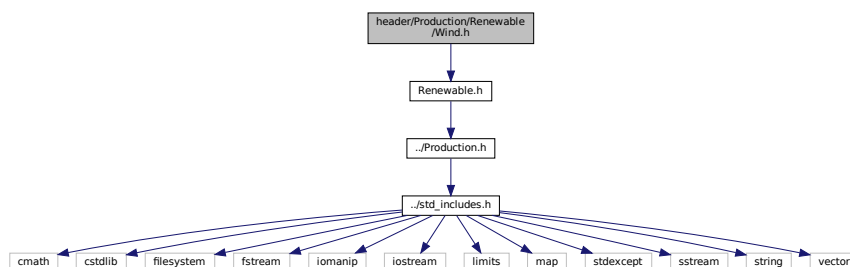
Header file the [Wave](#) class.

## 5.11 header/Production/Renewable/Wind.h File Reference

Header file the [Wind](#) class.

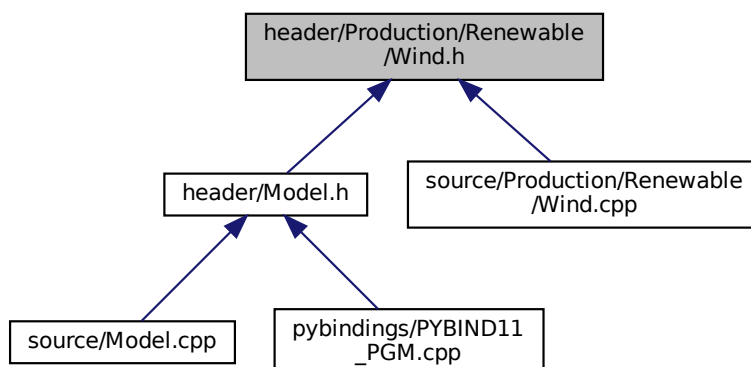
```
#include "Renewable.h"
```

Include dependency graph for `Wind.h`:





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



## Classes

- class [Wind](#)

*A derived class of the [Renewable](#) branch of [Production](#) which models wind production.*

### 5.11.1 Detailed Description

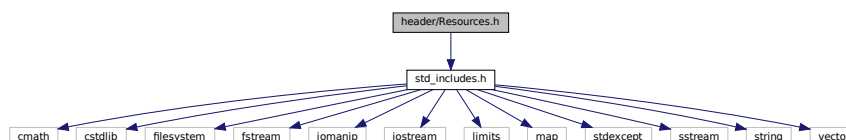
Header file the [Wind](#) class.

## 5.12 header/Resources.h File Reference

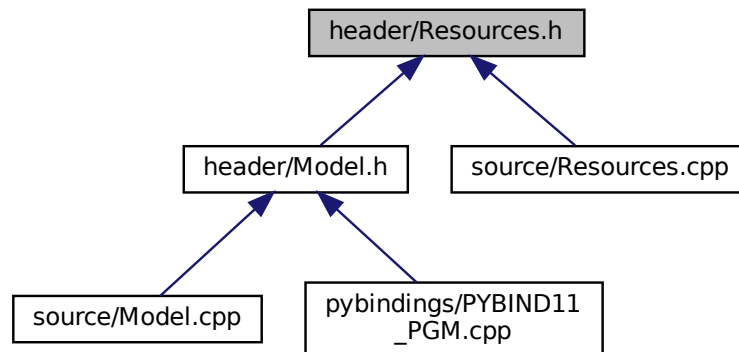
Header file the [Resources](#) class.

```
#include "std_includes.h"
```

Include dependency graph for Resources.h:



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



## Classes

- class [Resources](#)

*A class which contains renewable resource data. Intended to serve as a component class of [Model](#).*

### 5.12.1 Detailed Description

Header file the [Resources](#) class.

## 5.13 header/std\_includes.h File Reference

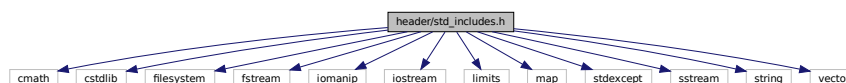
Header file which simply batches together the usual, standard includes.

```

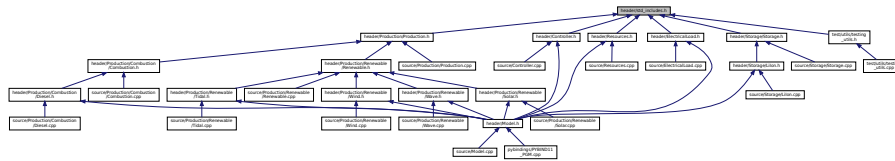
#include <cmath>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <limits>
#include <map>
#include <stdexcept>
#include <sstream>
#include <string>
#include <vector>

```

Include dependency graph for `std_includes.h`:



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



### 5.13.1 Detailed Description

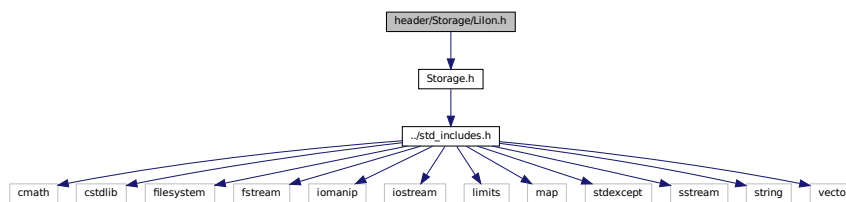
Header file which simply batches together the usual, standard includes.

## 5.14 header/Storage/Lilon.h File Reference

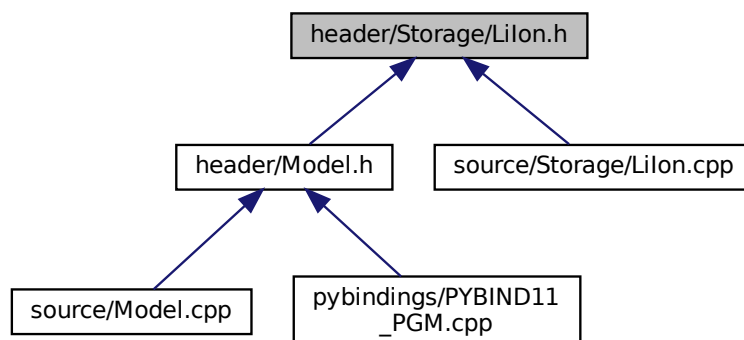
Header file the [Lilon](#) class.

```
#include "Storage.h"
```

Include dependency graph for Lilon.h:



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



## Classes

- class [Lilon](#)

A derived class of [Storage](#) which models energy storage by way of lithium-ion batteries.

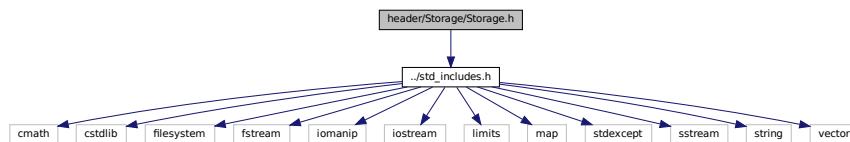
### 5.14.1 Detailed Description

Header file the [Lilon](#) class.

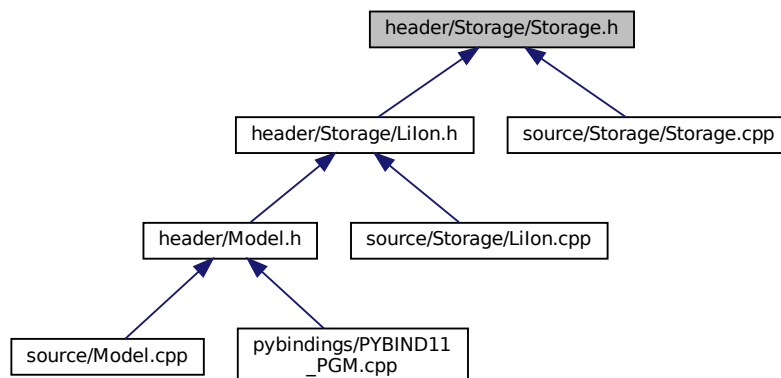
## 5.15 header/Storage/Storage.h File Reference

Header file the [Storage](#) class.

```
#include "../std_includes.h"
Include dependency graph for Storage.h:
```



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



## Classes

- class [Storage](#)

The base class of the [Storage](#) hierarchy. This hierarchy contains derived classes which model the storage of energy.

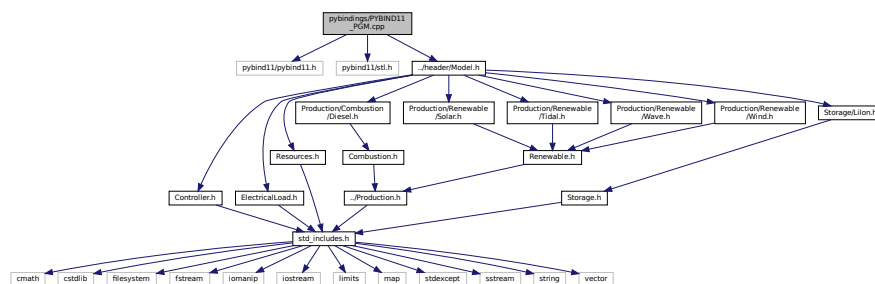
### 5.15.1 Detailed Description

Header file the [Storage](#) class.

## 5.16 pybindings/PYBIND11\_PGM.cpp File Reference

Python 3 bindings file for PGMcpp.

```
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>
#include "../header/Model.h"
Include dependency graph for PYBIND11_PGM.cpp:
```



## Functions

- [PYBIND11\\_MODULE](#) (PGMcpp, m)

### 5.16.1 Detailed Description

Python 3 bindings file for PGMcpp.

This is a file which defines the Python 3 bindings to be generated for PGMcpp. To generate bindings, use the provided setup.py.

ref: <https://pybind11.readthedocs.io/en/stable/>

### 5.16.2 Function Documentation

### 5.16.2.1 PYBIND11\_MODULE()

```

PYBIND11_MODULE (
    PGMcpp ,
    m )
{
30
31
32 // ===== Controller ===== //
33 /*
34 pybind11::class_<Controller>(m, "Controller")
35     .def(pybind11::init());
36 */
37 // ===== END Controller ===== //
38
39
40
41 // ===== ElectricalLoad ===== //
42 /*
43 pybind11::class_<ElectricalLoad>(m, "ElectricalLoad")
44     .def_readwrite("n_points", &ElectricalLoad::n_points)
45     .def_readwrite("max_load_kW", &ElectricalLoad::max_load_kW)
46     .def_readwrite("mean_load_kW", &ElectricalLoad::mean_load_kW)
47     .def_readwrite("min_load_kW", &ElectricalLoad::min_load_kW)
48     .def_readwrite("dt_vec_hrs", &ElectricalLoad::dt_vec_hrs)
49     .def_readwrite("load_vec_kW", &ElectricalLoad::load_vec_kW)
50     .def_readwrite("time_vec_hrs", &ElectricalLoad::time_vec_hrs)
51
52     .def(pybind11::init<std::string>());
53 */
54 // ===== END ElectricalLoad ===== //
55
56
57
58 // ===== Model ===== //
59 /*
60 pybind11::class_<Model>(m, "Model")
61     .def(
62         pybind11::init<
63             ElectricalLoad*,
64             RenewableResources*
65         >()
66     );
67 */
68 // ===== END Model ===== //
69
70
71
72 // ===== RenewableResources ===== //
73 /*
74 pybind11::class_<RenewableResources>(m, "RenewableResources")
75     .def(pybind11::init());
76     /*
77     .def(pybind11::init<>());
78     */
79 */
80 // ===== END RenewableResources ===== //
81
82 } /* PYBIND11_MODULE() */

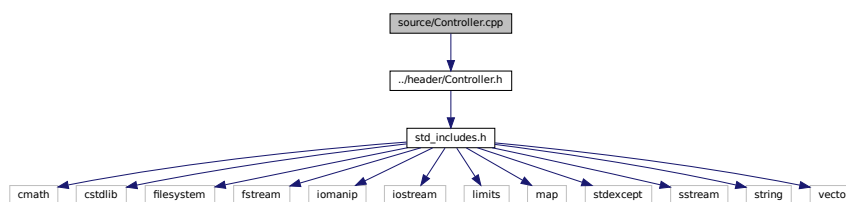
```

## 5.17 source/Controller.cpp File Reference

Implementation file for the [Controller](#) class.

```
#include "../header/Controller.h"
```

Include dependency graph for Controller.cpp:



### 5.17.1 Detailed Description

Implementation file for the [Controller](#) class.

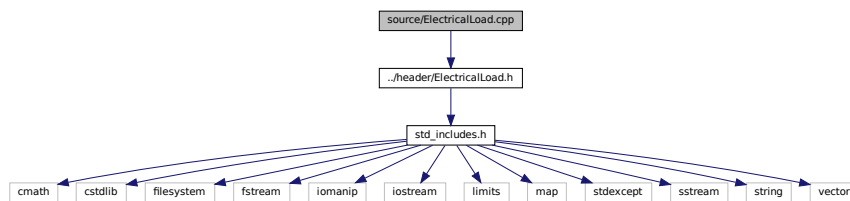
A class which contains a various dispatch control logic. Intended to serve as a component class of [Controller](#).

## 5.18 source/ElectricalLoad.cpp File Reference

Implementation file for the [ElectricalLoad](#) class.

```
#include "../header/ElectricalLoad.h"
```

Include dependency graph for ElectricalLoad.cpp:



### 5.18.1 Detailed Description

Implementation file for the [ElectricalLoad](#) class.

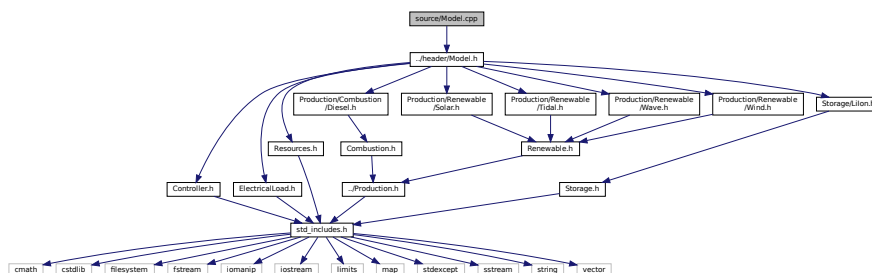
A class which contains time and electrical load data. Intended to serve as a component class of [Model](#).

## 5.19 source/Model.cpp File Reference

Implementation file for the [Model](#) class.

```
#include "../header/Model.h"
```

Include dependency graph for Model.cpp:



### 5.19.1 Detailed Description

Implementation file for the [Model](#) class.

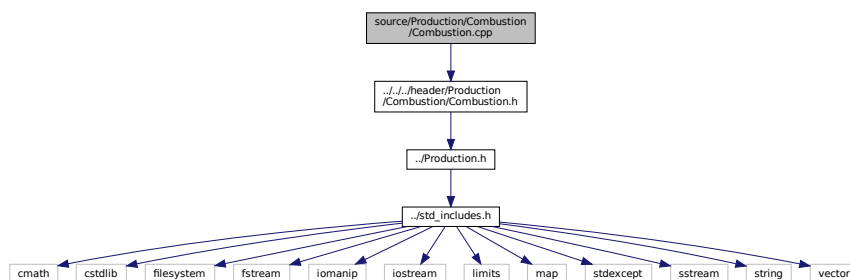
A container class which forms the centre of PGMcpp. The [Model](#) class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes.

## 5.20 source/Production/Combustion/Combustion.cpp File Reference

Implementation file for the [Combustion](#) class.

```
#include "../.../header/Production/Combustion/Combustion.h"
```

Include dependency graph for Combustion.cpp:



### 5.20.1 Detailed Description

Implementation file for the [Combustion](#) class.

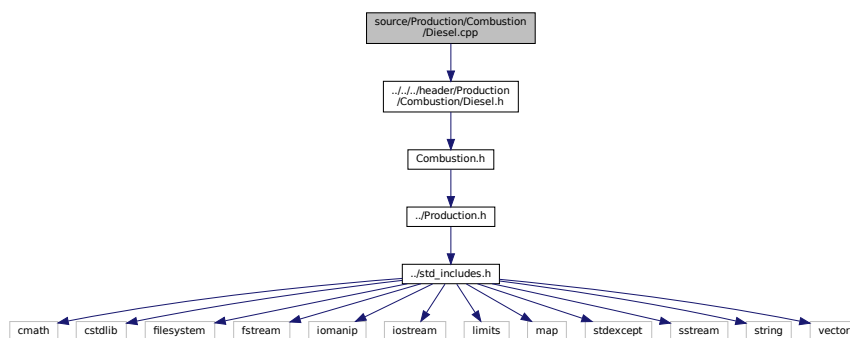
The root of the [Combustion](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

## 5.21 source/Production/Combustion/Diesel.cpp File Reference

Implementation file for the [Diesel](#) class.

```
#include "../.../header/Production/Combustion/Diesel.h"
```

Include dependency graph for Diesel.cpp:





### 5.21.1 Detailed Description

Implementation file for the [Diesel](#) class.

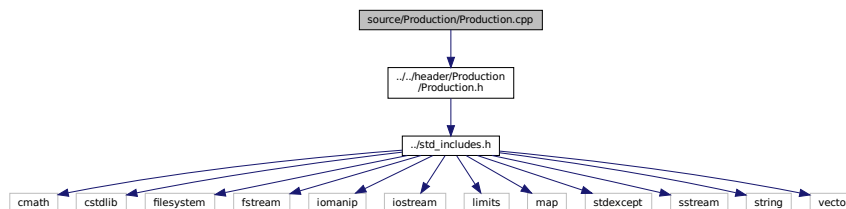
A derived class of the [Combustion](#) branch of [Production](#) which models production using a diesel generator.

## 5.22 source/Production/Production.cpp File Reference

Implementation file for the [Production](#) class.

```
#include "../..//header/Production/Production.h"
```

Include dependency graph for Production.cpp:



### 5.22.1 Detailed Description

Implementation file for the [Production](#) class.

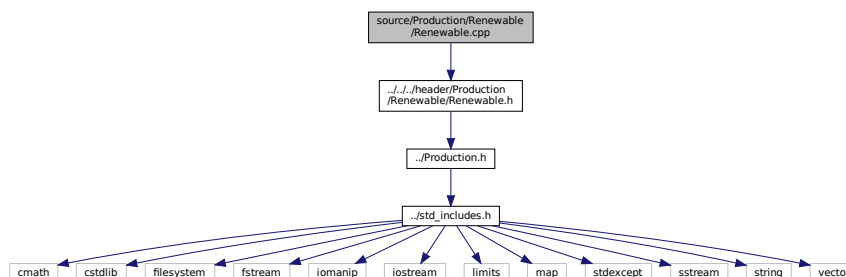
The base class of the [Production](#) hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise.

## 5.23 source/Production/Renewable/Renewable.cpp File Reference

Implementation file for the [Renewable](#) class.

```
#include "../..//..//header/Production/Renewable/Renewable.h"
```

Include dependency graph for Renewable.cpp:



### 5.23.1 Detailed Description

Implementation file for the [Renewable](#) class.

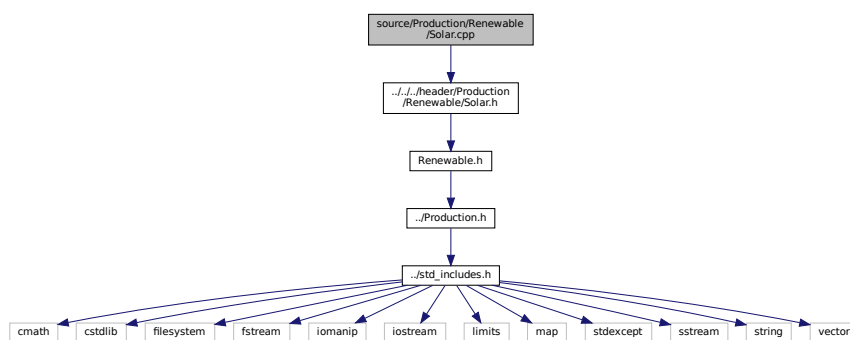
The root of the [Renewable](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the renewable production of energy.

## 5.24 source/Production/Renewable/Solar.cpp File Reference

Implementation file for the [Solar](#) class.

```
#include "../.../header/Production/Renewable/Solar.h"
```

Include dependency graph for Solar.cpp:



### 5.24.1 Detailed Description

Implementation file for the [Solar](#) class.

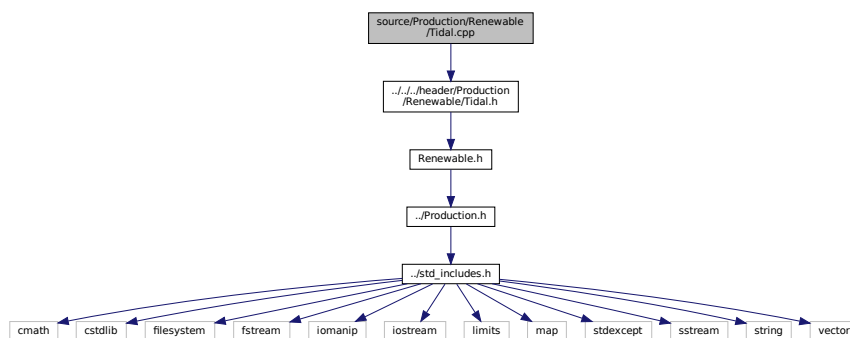
A derived class of the [Renewable](#) branch of [Production](#) which models solar production.

## 5.25 source/Production/Renewable/Tidal.cpp File Reference

Implementation file for the [Tidal](#) class.

```
#include "../.../header/Production/Renewable/Tidal.h"
```

Include dependency graph for Tidal.cpp:



### 5.25.1 Detailed Description

Implementation file for the [Tidal](#) class.

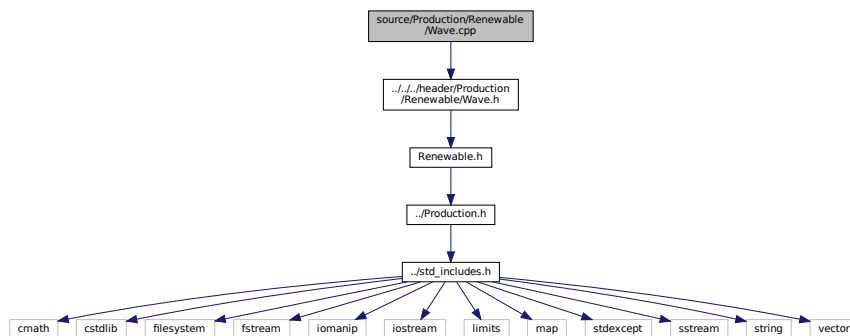
A derived class of the [Renewable](#) branch of [Production](#) which models tidal production.

## 5.26 source/Production/Renewable/Wave.cpp File Reference

Implementation file for the [Wave](#) class.

```
#include "../.../header/Production/Renewable/Wave.h"
```

Include dependency graph for Wave.cpp:



### 5.26.1 Detailed Description

Implementation file for the [Wave](#) class.

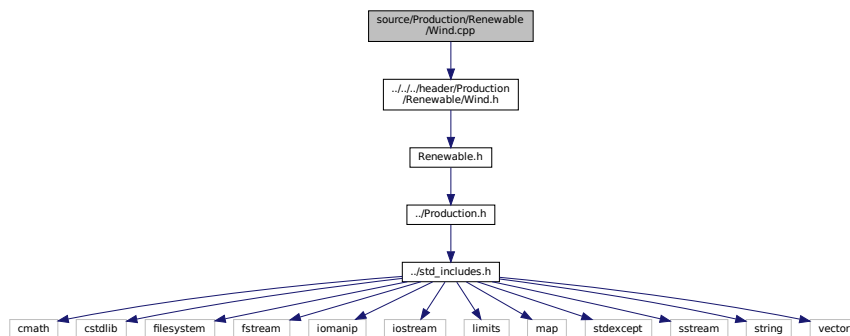
A derived class of the [Renewable](#) branch of [Production](#) which models wave production.

## 5.27 source/Production/Renewable/Wind.cpp File Reference

Implementation file for the [Wind](#) class.

```
#include "../.../header/Production/Renewable/Wind.h"
```

Include dependency graph for Wind.cpp:



### 5.27.1 Detailed Description

Implementation file for the [Wind](#) class.

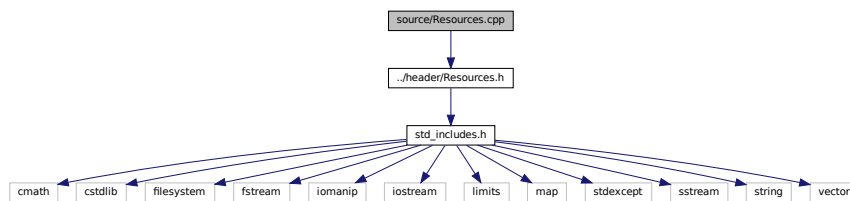
A derived class of the [Renewable](#) branch of [Production](#) which models wind production.

## 5.28 source/Resources.cpp File Reference

Implementation file for the [Resources](#) class.

```
#include "../header/Resources.h"
```

Include dependency graph for Resources.cpp:



### 5.28.1 Detailed Description

Implementation file for the [Resources](#) class.

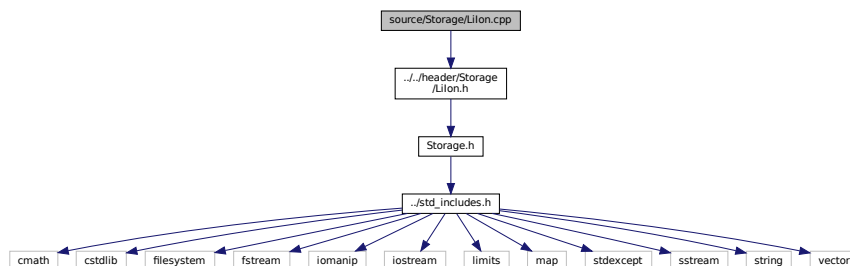
A class which contains renewable resource data. Intended to serve as a component class of [Model](#).

## 5.29 source/Storage/Lilon.cpp File Reference

Implementation file for the [Lilon](#) class.

```
#include "../../header/Storage/LiIon.h"
```

Include dependency graph for Lilon.cpp:



### 5.29.1 Detailed Description

Implementation file for the [Lilon](#) class.

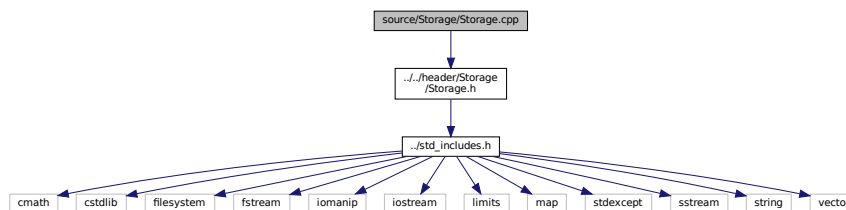
A derived class of [Storage](#) which models energy storage by way of lithium-ion batteries.

## 5.30 source/Storage/Storage.cpp File Reference

Implementation file for the [Storage](#) class.

```
#include "../..//header/Storage/Storage.h"
```

Include dependency graph for Storage.cpp:



### 5.30.1 Detailed Description

Implementation file for the [Storage](#) class.

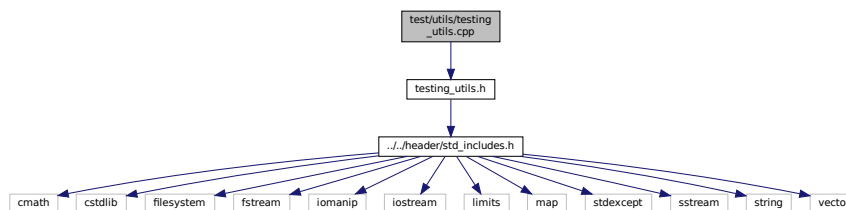
The base class of the [Storage](#) hierarchy. This hierarchy contains derived classes which model the storage of energy.

## 5.31 test/utis/testing\_utils.cpp File Reference

Header file for various PGMcpp testing utilities.

```
#include "testing_utils.h"
```

Include dependency graph for testing\_utils.cpp:



## Functions

- void `printGreen` (std::string input\_str)  
A function that sends green text to std::cout.
- void `printGold` (std::string input\_str)  
A function that sends gold text to std::cout.
- void `printRed` (std::string input\_str)  
A function that sends red text to std::cout.
- void `testFloatEquals` (double x, double y, std::string file, int line)  
Tests for the equality of two floating point numbers x and y (to within `FLOAT_TOLERANCE`).
- void `testGreaterThan` (double x, double y, std::string file, int line)  
Tests if  $x > y$ .
- void `testGreaterThanOrEqualTo` (double x, double y, std::string file, int line)  
Tests if  $x \geq y$ .
- void `testLessThan` (double x, double y, std::string file, int line)  
Tests if  $x < y$ .
- void `testLessThanOrEqualTo` (double x, double y, std::string file, int line)  
Tests if  $x \leq y$ .
- void `testTruth` (bool statement, std::string file, int line)  
Tests if the given statement is true.
- void `expectedErrorNotDetected` (std::string file, int line)  
A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

### 5.31.1 Detailed Description

Header file for various PGMcpp testing utilities.

This is a library of utility functions used throughout the various test suites.

### 5.31.2 Function Documentation

#### 5.31.2.1 `expectedErrorNotDetected()`

```
void expectedErrorNotDetected (
    std::string file,
    int line )
```

A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

#### Parameters

<i>file</i>	The file in which the test is applied (you should be able to just pass in " <code>__FILE__</code> ").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in " <code>__LINE__</code> ").

```
432 {
433     std::string error_str = "\n ERROR   failed to throw expected error prior to line ";
434     error_str += std::to_string(line);
```

```
435     error_str += " of ";
436     error_str += file;
437
438     #ifdef _WIN32
439         std::cout << error_str << std::endl;
440     #endif
441
442     throw std::runtime_error(error_str);
443     return;
444 } /* expectedErrorNotDetected() */
```

### 5.31.2.2 printGold()

```
void printGold (
    std::string input_str )
```

A function that sends gold text to std::cout.

#### Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```
84 {
85     std::cout << "\x1B[33m" << input_str << "\033[0m";
86     return;
87 } /* printGold() */
```

### 5.31.2.3 printGreen()

```
void printGreen (
    std::string input_str )
```

A function that sends green text to std::cout.

#### Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```
64 {
65     std::cout << "\x1B[32m" << input_str << "\033[0m";
66     return;
67 } /* printGreen() */
```

### 5.31.2.4 printRed()

```
void printRed (
    std::string input_str )
```

A function that sends red text to std::cout.

**Parameters**

<i>input_str</i>	The text of the string to be sent to <code>std::cout</code> .
------------------	---

```

104 {
105     std::cout << "\x1B[31m" << input_str << "\033[0m";
106     return;
107 } /* printRed() */

```

**5.31.2.5 testFloatEquals()**

```

void testFloatEquals (
    double x,
    double y,
    std::string file,
    int line )

```

Tests for the equality of two floating point numbers  $x$  and  $y$  (to within `FLOAT_TOLERANCE`).

**Parameters**

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in " <code>__FILE__</code> ").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in " <code>__LINE__</code> ").

```

138 {
139     if (fabs(x - y) <= FLOAT_TOLERANCE) {
140         return;
141     }
142
143     std::string error_str = "ERROR: testFloatEquals():\t in ";
144     error_str += file;
145     error_str += "\tline ";
146     error_str += std::to_string(line);
147     error_str += ":\t\n";
148     error_str += std::to_string(x);
149     error_str += " and ";
150     error_str += std::to_string(y);
151     error_str += " are not equal to within +/- ";
152     error_str += std::to_string(FLOAT_TOLERANCE);
153     error_str += "\n";
154
155     #ifdef WIN32
156         std::cout << error_str << std::endl;
157     #endif
158
159     throw std::runtime_error(error_str);
160     return;
161 } /* testFloatEquals() */

```

**5.31.2.6 testGreaterThan()**

```

void testGreaterThan (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x > y$ .



## Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

191 {
192     if (x > y) {
193         return;
194     }
195
196     std::string error_str = "ERROR: testGreaterThan():\t in ";
197     error_str += file;
198     error_str += "\tline ";
199     error_str += std::to_string(line);
200     error_str += ":\t\n";
201     error_str += std::to_string(x);
202     error_str += " is not greater than ";
203     error_str += std::to_string(y);
204     error_str += "\n";
205
206     #ifdef _WIN32
207         std::cout << error_str << std::endl;
208     #endif
209
210     throw std::runtime_error(error_str);
211     return;
212 } /* testGreaterThan() */

```

## 5.31.2.7 testGreaterThanOrEqualTo()

```

void testGreaterThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x \geq y$ .

## Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

242 {
243     if (x >= y) {
244         return;
245     }
246
247     std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
248     error_str += file;
249     error_str += "\tline ";
250     error_str += std::to_string(line);
251     error_str += ":\t\n";
252     error_str += std::to_string(x);
253     error_str += " is not greater than or equal to ";
254     error_str += std::to_string(y);
255     error_str += "\n";
256
257     #ifdef _WIN32
258         std::cout << error_str << std::endl;
259     #endif
260
261     throw std::runtime_error(error_str);

```

```

262     return;
263 } /* testGreaterThanOrEqualTo() */

```

### 5.31.2.8 testLessThan()

```

void testLessThan (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x < y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

293 {
294     if (x < y) {
295         return;
296     }
297
298     std::string error_str = "ERROR: testLessThan():\t in ";
299     error_str += file;
300     error_str += "\tline ";
301     error_str += std::to_string(line);
302     error_str += ":\t\n";
303     error_str += std::to_string(x);
304     error_str += " is not less than ";
305     error_str += std::to_string(y);
306     error_str += "\n";
307
308     #ifdef _WIN32
309         std::cout << error_str << std::endl;
310     #endif
311
312     throw std::runtime_error(error_str);
313     return;
314 } /* testLessThan() */

```

### 5.31.2.9 testLessThanOrEqualTo()

```

void testLessThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x \leq y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

344 {
345     if (x <= y) {
346         return;
347     }
348
349     std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
350     error_str += file;
351     error_str += "\tline ";
352     error_str += std::to_string(line);
353     error_str += ":\t\n";
354     error_str += std::to_string(x);
355     error_str += " is not less than or equal to ";
356     error_str += std::to_string(y);
357     error_str += "\n";
358
359     #ifdef _WIN32
360         std::cout << error_str << std::endl;
361     #endif
362
363     throw std::runtime_error(error_str);
364     return;
365 } /* testLessThanOrEqualTo() */

```

### 5.31.2.10 testTruth()

```

void testTruth (
    bool statement,
    std::string file,
    int line )

```

Tests if the given statement is true.

#### Parameters

<i>statement</i>	The statement whose truth is to be tested ("1 == 0", for example).
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

392 {
393     if (statement) {
394         return;
395     }
396
397     std::string error_str = "ERROR: testTruth():\t in ";
398     error_str += file;
399     error_str += "\tline ";
400     error_str += std::to_string(line);
401     error_str += ":\t\n";
402     error_str += "Given statement is not true";
403
404     #ifdef _WIN32
405         std::cout << error_str << std::endl;
406     #endif
407
408     throw std::runtime_error(error_str);
409     return;
410 } /* testTruth() */

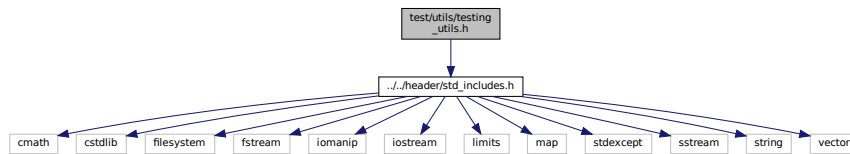
```

## 5.32 test/utills/testing\_utils.h File Reference

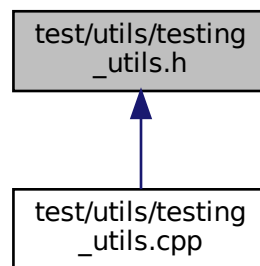
Header file for various PGMcpp testing utilities.

```
#include "../..//header/std_includes.h"
```

Include dependency graph for testing\_utils.h:



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



## Macros

- `#define FLOAT\_TOLERANCE 1e-6`  
A tolerance for application to floating point equality tests.

## Functions

- void [printGreen](#) (std::string)  
A function that sends green text to `std::cout`.
- void [printGold](#) (std::string)  
A function that sends gold text to `std::cout`.
- void [printRed](#) (std::string)  
A function that sends red text to `std::cout`.
- void [testFloatEquals](#) (double, double, std::string, int)  
Tests for the equality of two floating point numbers  $x$  and  $y$  (to within `FLOAT_TOLERANCE`).
- void [testGreaterThan](#) (double, double, std::string, int)  
Tests if  $x > y$ .
- void [testGreaterThanOrEqualTo](#) (double, double, std::string, int)  
Tests if  $x \geq y$ .
- void [testLessThan](#) (double, double, std::string, int)  
Tests if  $x < y$ .

- void `testLessThanOrEqualTo` (double, double, std::string, int)  
*Tests if  $x \leq y$ .*
- void `testTruth` (bool, std::string, int)  
*Tests if the given statement is true.*
- void `expectedErrorNotDetected` (std::string, int)  
*A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.*

### 5.32.1 Detailed Description

Header file for various PGMcpp testing utilities.

This is a library of utility functions used throughout the various test suites.

### 5.32.2 Macro Definition Documentation

#### 5.32.2.1 FLOAT\_TOLERANCE

```
#define FLOAT_TOLERANCE 1e-6
```

A tolerance for application to floating point equality tests.

### 5.32.3 Function Documentation

#### 5.32.3.1 expectedErrorNotDetected()

```
void expectedErrorNotDetected (
    std::string file,
    int line )
```

A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

#### Parameters

<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
432 {
433     std::string error_str = "\n ERROR   failed to throw expected error prior to line ";
434     error_str += std::to_string(line);
435     error_str += " of ";
436     error_str += file;
437
438     #ifdef _WIN32
439         std::cout << error_str << std::endl;
440     #endif
441 }
```

```

442     throw std::runtime_error(error_str);
443     return;
444 } /* expectedErrorNotDetected() */

```

### 5.32.3.2 printGold()

```

void printGold (
    std::string input_str )

```

A function that sends gold text to std::cout.

#### Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

84 {
85     std::cout << "\x1B[33m" << input_str << "\033[0m";
86     return;
87 } /* printGold() */

```

### 5.32.3.3 printGreen()

```

void printGreen (
    std::string input_str )

```

A function that sends green text to std::cout.

#### Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

64 {
65     std::cout << "\x1B[32m" << input_str << "\033[0m";
66     return;
67 } /* printGreen() */

```

### 5.32.3.4 printRed()

```

void printRed (
    std::string input_str )

```

A function that sends red text to std::cout.

#### Parameters

<i>input_str</i>	The text of the string to be sent to std::cout.
------------------	---

```

104 {
105     std::cout << "\x1B[31m" << input_str << "\033[0m";

```

```

106     return;
107 } /* printRed() */

```

### 5.32.3.5 testFloatEquals()

```

void testFloatEquals (
    double x,
    double y,
    std::string file,
    int line )

```

Tests for the equality of two floating point numbers  $x$  and  $y$  (to within `FLOAT_TOLERANCE`).

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in " <code>__FILE__</code> ").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in " <code>__LINE__</code> ").

```

138 {
139     if (fabs(x - y) <= FLOAT_TOLERANCE) {
140         return;
141     }
142
143     std::string error_str = "ERROR: testFloatEquals():\t in ";
144     error_str += file;
145     error_str += "\tline ";
146     error_str += std::to_string(line);
147     error_str += ":\t\n";
148     error_str += std::to_string(x);
149     error_str += " and ";
150     error_str += std::to_string(y);
151     error_str += " are not equal to within +/- ";
152     error_str += std::to_string(FLOAT_TOLERANCE);
153     error_str += "\n";
154
155     #ifdef _WIN32
156         std::cout << error_str << std::endl;
157     #endif
158
159     throw std::runtime_error(error_str);
160     return;
161 } /* testFloatEquals() */

```

### 5.32.3.6 testGreaterThan()

```

void testGreaterThan (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x > y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
----------	-----------------------------------

## Parameters

<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

191 {
192     if (x > y) {
193         return;
194     }
195
196     std::string error_str = "ERROR: testGreaterThan():\t in ";
197     error_str += file;
198     error_str += "\tline ";
199     error_str += std::to_string(line);
200     error_str += ":\t\n";
201     error_str += std::to_string(x);
202     error_str += " is not greater than ";
203     error_str += std::to_string(y);
204     error_str += "\n";
205
206     #ifdef _WIN32
207         std::cout << error_str << std::endl;
208     #endif
209
210     throw std::runtime_error(error_str);
211     return;
212 } /* testGreaterThan() */

```

## 5.32.3.7 testGreaterThanOrEqualTo()

```

void testGreaterThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )

```

Tests if  $x \geq y$ .

## Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

242 {
243     if (x >= y) {
244         return;
245     }
246
247     std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
248     error_str += file;
249     error_str += "\tline ";
250     error_str += std::to_string(line);
251     error_str += ":\t\n";
252     error_str += std::to_string(x);
253     error_str += " is not greater than or equal to ";
254     error_str += std::to_string(y);
255     error_str += "\n";
256
257     #ifdef _WIN32
258         std::cout << error_str << std::endl;
259     #endif
260
261     throw std::runtime_error(error_str);
262     return;

```



```
263 }    /* testGreaterThanOrEqualTo() */
```

### 5.32.3.8 testLessThan()

```
void testLessThan (
    double x,
    double y,
    std::string file,
    int line )
```

Tests if  $x < y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
293 {
294     if (x < y) {
295         return;
296     }
297
298     std::string error_str = "ERROR: testLessThan():\t in ";
299     error_str += file;
300     error_str += "\tline ";
301     error_str += std::to_string(line);
302     error_str += ":\t\n";
303     error_str += std::to_string(x);
304     error_str += " is not less than ";
305     error_str += std::to_string(y);
306     error_str += "\n";
307
308     #ifdef _WIN32
309         std::cout << error_str << std::endl;
310     #endif
311
312     throw std::runtime_error(error_str);
313     return;
314 }    /* testLessThan() */
```

### 5.32.3.9 testLessThanOrEqualTo()

```
void testLessThanOrEqualTo (
    double x,
    double y,
    std::string file,
    int line )
```

Tests if  $x \leq y$ .

#### Parameters

<i>x</i>	The first of two numbers to test.
<i>y</i>	The second of two numbers to test.
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in " __LINE__ ").

```

344 {
345     if (x <= y) {
346         return;
347     }
348
349     std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
350     error_str += file;
351     error_str += "\tline ";
352     error_str += std::to_string(line);
353     error_str += ":\t\n";
354     error_str += std::to_string(x);
355     error_str += " is not less than or equal to ";
356     error_str += std::to_string(y);
357     error_str += "\n";
358
359     #ifdef _WIN32
360         std::cout << error_str << std::endl;
361     #endif
362
363     throw std::runtime_error(error_str);
364     return;
365 } /* testLessThanOrEqualTo() */

```

### 5.32.3.10 testTruth()

```

void testTruth (
    bool statement,
    std::string file,
    int line )

```

Tests if the given statement is true.

#### Parameters

<i>statement</i>	The statement whose truth is to be tested ("1 == 0", for example).
<i>file</i>	The file in which the test is applied (you should be able to just pass in "__FILE__").
<i>line</i>	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```

392 {
393     if (statement) {
394         return;
395     }
396
397     std::string error_str = "ERROR: testTruth():\t in ";
398     error_str += file;
399     error_str += "\tline ";
400     error_str += std::to_string(line);
401     error_str += ":\t\n";
402     error_str += "Given statement is not true";
403
404     #ifdef _WIN32
405         std::cout << error_str << std::endl;
406     #endif
407
408     throw std::runtime_error(error_str);
409     return;
410 } /* testTruth() */

```

# Index

- ~Combustion
  - Combustion, [8](#)
- ~Controller
  - Controller, [9](#)
- ~Diesel
  - Diesel, [11](#)
- ~ElectricalLoad
  - ElectricalLoad, [12](#)
- ~Lilon
  - Lilon, [14](#)
- ~Model
  - Model, [16](#)
- ~Production
  - Production, [18](#)
- ~Renewable
  - Renewable, [20](#)
- ~Resources
  - Resources, [21](#)
- ~Solar
  - Solar, [23](#)
- ~Storage
  - Storage, [24](#)
- ~Tidal
  - Tidal, [26](#)
- ~Wave
  - Wave, [28](#)
- ~Wind
  - Wind, [30](#)
- Combustion, [7](#)
  - ~Combustion, [8](#)
  - Combustion, [8](#)
- combustion\_ptr\_vec
  - Model, [16](#)
- Controller, [9](#)
  - ~Controller, [9](#)
  - Controller, [9](#)
- controller
  - Model, [16](#)
- Diesel, [10](#)
  - ~Diesel, [11](#)
  - Diesel, [11](#)
- electrical\_load
  - Model, [16](#)
- ElectricalLoad, [12](#)
  - ~ElectricalLoad, [12](#)
  - ElectricalLoad, [12](#)
- expectedErrorNotDetected
  - testing\_utils.cpp, [54](#)
  - testing\_utils.h, [61](#)
- FLOAT\_TOLERANCE
  - testing\_utils.h, [61](#)
- header/Controller.h, [31](#)
- header/ElectricalLoad.h, [32](#)
- header/Model.h, [33](#)
- header/Production/Combustion/Combustion.h, [34](#)
- header/Production/Combustion/Diesel.h, [35](#)
- header/Production/Production.h, [36](#)
- header/Production/Renewable/Renewable.h, [36](#)
- header/Production/Renewable/Solar.h, [37](#)
- header/Production/Renewable/Tidal.h, [38](#)
- header/Production/Renewable/Wave.h, [39](#)
- header/Production/Renewable/Wind.h, [40](#)
- header/Resources.h, [41](#)
- header/std\_includes.h, [42](#)
- header/Storage/Lilon.h, [43](#)
- header/Storage/Storage.h, [44](#)
- Lilon, [13](#)
  - ~Lilon, [14](#)
  - Lilon, [14](#)
- Model, [15](#)
  - ~Model, [16](#)
  - combustion\_ptr\_vec, [16](#)
  - controller, [16](#)
  - electrical\_load, [16](#)
  - Model, [16](#)
  - renewable\_ptr\_vec, [17](#)
  - resources, [17](#)
  - storage\_ptr\_vec, [17](#)
- printGold
  - testing\_utils.cpp, [55](#)
  - testing\_utils.h, [62](#)
- printGreen
  - testing\_utils.cpp, [55](#)
  - testing\_utils.h, [62](#)
- printRed
  - testing\_utils.cpp, [55](#)
  - testing\_utils.h, [62](#)
- Production, [17](#)
  - ~Production, [18](#)
  - Production, [18](#)
- PYBIND11\_MODULE
  - PYBIND11\_PGM.cpp, [45](#)
- PYBIND11\_PGM.cpp

- PYBIND11\_MODULE, [45](#)
- pybindings/PYBIND11\_PGM.cpp, [45](#)
- Renewable, [19](#)
  - ~Renewable, [20](#)
  - Renewable, [20](#)
- renewable\_ptr\_vec
  - Model, [17](#)
- Resources, [20](#)
  - ~Resources, [21](#)
  - Resources, [21](#)
- resources
  - Model, [17](#)
- Solar, [21](#)
  - ~Solar, [23](#)
  - Solar, [23](#)
- source/Controller.cpp, [46](#)
- source/ElectricalLoad.cpp, [47](#)
- source/Model.cpp, [47](#)
- source/Production/Combustion/Combustion.cpp, [48](#)
- source/Production/Combustion/Diesel.cpp, [48](#)
- source/Production/Production.cpp, [49](#)
- source/Production/Renewable/Renewable.cpp, [49](#)
- source/Production/Renewable/Solar.cpp, [50](#)
- source/Production/Renewable/Tidal.cpp, [50](#)
- source/Production/Renewable/Wave.cpp, [51](#)
- source/Production/Renewable/Wind.cpp, [51](#)
- source/Resources.cpp, [52](#)
- source/Storage/Lilon.cpp, [52](#)
- source/Storage/Storage.cpp, [53](#)
- Storage, [23](#)
  - ~Storage, [24](#)
  - Storage, [24](#)
- storage\_ptr\_vec
  - Model, [17](#)
- test/utls/testing\_utils.cpp, [53](#)
- test/utls/testing\_utils.h, [59](#)
- testFloatEquals
  - testing\_utils.cpp, [56](#)
  - testing\_utils.h, [63](#)
- testGreaterThan
  - testing\_utils.cpp, [56](#)
  - testing\_utils.h, [63](#)
- testGreaterThanOrEqualTo
  - testing\_utils.cpp, [57](#)
  - testing\_utils.h, [64](#)
- testing\_utils.cpp
  - expectedErrorNotDetected, [54](#)
  - printGold, [55](#)
  - printGreen, [55](#)
  - printRed, [55](#)
  - testFloatEquals, [56](#)
  - testGreaterThan, [56](#)
  - testGreaterThanOrEqualTo, [57](#)
  - testLessThan, [58](#)
  - testLessThanOrEqualTo, [58](#)
  - testTruth, [59](#)
- testing\_utils.h
  - expectedErrorNotDetected, [61](#)
  - FLOAT\_TOLERANCE, [61](#)
  - printGold, [62](#)
  - printGreen, [62](#)
  - printRed, [62](#)
  - testFloatEquals, [63](#)
  - testGreaterThan, [63](#)
  - testGreaterThanOrEqualTo, [64](#)
  - testLessThan, [65](#)
  - testLessThanOrEqualTo, [65](#)
  - testTruth, [66](#)
- testLessThan
  - testing\_utils.cpp, [58](#)
  - testing\_utils.h, [65](#)
- testLessThanOrEqualTo
  - testing\_utils.cpp, [58](#)
  - testing\_utils.h, [65](#)
- testTruth
  - testing\_utils.cpp, [59](#)
  - testing\_utils.h, [66](#)
- Tidal, [25](#)
  - ~Tidal, [26](#)
  - Tidal, [26](#)
- Wave, [27](#)
  - ~Wave, [28](#)
  - Wave, [28](#)
- Wind, [29](#)
  - ~Wind, [30](#)
  - Wind, [30](#)