PGMcpp: PRIMED Grid Modelling (in C++)

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

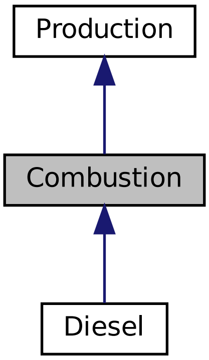
Class Documentation

4.1 Combustion Class Reference

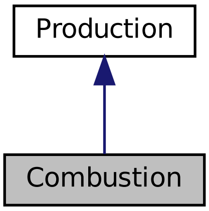
The root of the Combustion branch of the Production hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

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

Inheritance diagram for Combustion:



Collaboration diagram for Combustion:



Public Member Functions

- Combustion (void)

  Constructor (dummy) for the Combustion class.
- Combustion (int, CombustionInputs)

  Constructor (intended) for the Combustion class.
- virtual double requestProductionkW (int, double, double)
- virtual double commit (int, double, double, double)

  Method which takes in production and load for the current timestep, computes and records dispatch and curtailment, and then returns remaining load.
- double getFuelConsumptionL (double, double)

  Method which takes in production and returns volume of fuel burned over the given interval of time.
- Emissions getEmissionskg (double)

  Method which takes in volume of fuel consumed and returns mass spectrum of resulting emissions.
- virtual ∼Combustion (void)

  Destructor for the Combustion class.

Public Attributes

- double linear_fuel_slope_LkWh

  The slope [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced.
- double linear_fuel_intercept_LkWh

  The intercept [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced.
- std::vector< double > fuel_consumption_vec_L

  A vector of fuel consumed [L] over each modelling time step.
- std::vector< double > fuel_cost_vec

  A vector of fuel costs (undefined currency) incurred over each modelling time step. These costs are not discounted (i.e., these are nominal costs).
- std::vector< double > CO2_emissions_vec_kg

  A vector of carbon dioxide (CO2) emitted [kg] over each modelling time step.
- std::vector< double > CO_emissions_vec_kg

  A vector of carbon monoxide (CO) emitted [kg] over each modelling time step.
- std::vector< double > NOx_emissions_vec_kg

  A vector of nitrogen oxide (NOx) emitted [kg] over each modelling time step.

- std::vector< double > SOx_emissions_vec_kg

  A vector of sulfur oxide (SOx) emitted [kg] over each modelling time step.
- std::vector< double > CH4_emissions_vec_kg

  A vector of methane (CH4) emitted [kg] over each modelling time step.
- std::vector< double > PM_emissions_vec_kg

  A vector of particulate matter (PM) emitted [kg] over each modelling time step.

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() [1/2]

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

Constructor (dummy) for the Combustion class.

```
{
    return;
}   /* Combustion() */
```

4.1.2.2 Combustion() [2/2]

```
Combustion::Combustion (
            int n_points,
            CombustionInputs combustion_inputs )
```

Constructor (intended) for the Combustion class.

Parameters

n_points	The number of points in the modelling time series.
combustion_inputs	A structure of Combustion constructor inputs.

```
                                                                              :
Production(n_points, combustion_inputs.production_inputs)
{
    //  1. check inputs
    this->__checkInputs(combustion_inputs);

    //  2. set attributes
    this->linear_fuel_slope_LkWh = 0;
    this->linear_fuel_intercept_LkWh = 0;

    this->fuel_consumption_vec_L.resize(this->n_points, 0);
    this->fuel_cost_vec.resize(this->n_points, 0);

    this->CO2_emissions_vec_kg.resize(this->n_points, 0);
    this->CO_emissions_vec_kg.resize(this->n_points, 0);
    this->NOx_emissions_vec_kg.resize(this->n_points, 0);
    this->SOx_emissions_vec_kg.resize(this->n_points, 0);
    this->CH4_emissions_vec_kg.resize(this->n_points, 0);
    this->PM_emissions_vec_kg.resize(this->n_points, 0);

    //  3. construction print
    if (this->print_flag) {
        std::cout « "Combustion object constructed at " « this « std::endl;
    }

    return;
}   /* Combustion() */
```

4.1.2.3 ∼Combustion()

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

Destructor for the Combustion class.

```
{
    //  1. destruction print
    if (this->print_flag) {
        std::cout « "Combustion object at " « this « " destroyed" « std::endl;
    }

    return;
}   /* ~Combustion() */
```

4.1.3 Member Function Documentation

4.1.3.1 commit()

```
double Combustion::commit (
              int timestep,
              double dt_hrs,
              double production_kW,
              double load_kW )  [virtual]
```

Method which takes in production and load for the current timestep, computes and records dispatch and curtailment, and then returns remaining load.

Parameters

timestep	The timestep (i.e., time series index) for the request.
dt_hrs	The interval of time [hrs] associated with the timestep.
production_kW	The production [kW] of the asset in this timestep.
load_kW	The load [kW] passed to the asset in this timestep.

Returns

load_kW The load [kW] remaining after the dispatch is deducted from it.

Reimplemented from Production.

Reimplemented in Diesel.

```
{
    //  1. invoke base class method
    load_kW = Production :: commit(
        timestep,
        dt_hrs,
        production_kW,
        load_kW
    );


    if (this->is_running) {
        //  2. compute and record fuel consumption
        double fuel_consumed_L = this->getFuelConsumptionL(dt_hrs, production_kW);
        this->fuel_consumption_vec_L[timestep] = fuel_consumed_L;

        //  3. compute and record emissions
        //...

        //  4. incur fuel costs
        //...
    }

    return load_kW;
}
```

4.1.3.2 getEmissionskg()

```
Emissions Combustion::getEmissionskg (
              double fuel_consumed_L )
```

Method which takes in volume of fuel consumed and returns mass spectrum of resulting emissions.

Parameters

fuel_consumed←↩_L	The volume of fuel consumed [L].

Returns

Emissions A structure containing the mass spectrum of resulting emissions.

```
                                                              {
    Emissions emissions;

    //...

    return emissions;
}   /* getEmissionskg() */
```

4.1.3.3 getFuelConsumptionL()

```
double Combustion::getFuelConsumptionL (
              double dt_hrs,
              double production_kW )
```

Method which takes in production and returns volume of fuel burned over the given interval of time.

Parameters

dt_hrs	The interval of time [hrs] associated with the timestep.
production_kW	The production [kW] of the asset in this timestep.

```
{
    double fuel_consumed_L = (
        this->linear_fuel_slope_LkWh * production_kW +
        this->linear_fuel_intercept_LkWh * this->capacity_kW
    ) * dt_hrs;

    return fuel_consumed_L;
} /* getFuelConsumption() */
```

4.1.3.4 requestProductionkW()

```
virtual double Combustion::requestProductionkW (
            int ,
            double ,
            double  )  [inline], [virtual]
```

Reimplemented in Diesel.

```
{return 0;}
```

4.1.4 Member Data Documentation

4.1.4.1 CH4_emissions_vec_kg

```
std::vector<double> Combustion::CH4_emissions_vec_kg
```

A vector of methane (CH4) emitted [kg] over each modelling time step.

4.1.4.2 CO2_emissions_vec_kg

```
std::vector<double> Combustion::CO2_emissions_vec_kg
```

A vector of carbon dioxide (CO2) emitted [kg] over each modelling time step.

4.1.4.3 CO_emissions_vec_kg

```
std::vector<double> Combustion::CO_emissions_vec_kg
```

A vector of carbon monoxide (CO) emitted [kg] over each modelling time step.

4.1.4.4 fuel_consumption_vec_L

```
std::vector<double> Combustion::fuel_consumption_vec_L
```

A vector of fuel consumed [L] over each modelling time step.

4.1.4.5 fuel_cost_vec

```
std::vector<double> Combustion::fuel_cost_vec
```

A vector of fuel costs (undefined currency) incurred over each modelling time step. These costs are not discounted (i.e., these are nominal costs).

4.1.4.6 linear_fuel_intercept_LkWh

```
double Combustion::linear_fuel_intercept_LkWh
```

The intercept [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced.

4.1.4.7 linear_fuel_slope_LkWh

```
double Combustion::linear_fuel_slope_LkWh
```

The slope [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced.

4.1.4.8 NOx_emissions_vec_kg

```
std::vector<double> Combustion::NOx_emissions_vec_kg
```

A vector of nitrogen oxide (NOx) emitted [kg] over each modelling time step.

4.1.4.9 PM_emissions_vec_kg

```
std::vector<double> Combustion::PM_emissions_vec_kg
```

A vector of particulate matter (PM) emitted [kg] over each modelling time step.

4.1.4.10 SOx_emissions_vec_kg

```
std::vector<double> Combustion::SOx_emissions_vec_kg
```

A vector of sulfur oxide (SOx) emitted [kg] over each modelling time step.

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

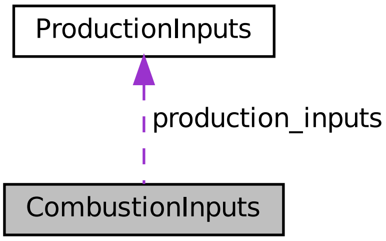
- header/Production/Combustion/Combustion.h
- source/Production/Combustion/Combustion.cpp

4.2 CombustionInputs Struct Reference

A structure which bundles the necessary inputs for the Combustion constructor. Provides default values for every necessary input. Note that this structure encapsulates ProductionInputs.

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

Collaboration diagram for CombustionInputs:



Public Attributes

- ProductionInputs production_inputs

    An encapsulated ProductionInputs instance.

4.2.1 Detailed Description

A structure which bundles the necessary inputs for the Combustion constructor. Provides default values for every necessary input. Note that this structure encapsulates ProductionInputs.

4.2.2 Member Data Documentation

4.2.2.1 production_inputs

```
ProductionInputs CombustionInputs::production_inputs
```

An encapsulated ProductionInputs instance.

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

- header/Production/Combustion/Combustion.h

4.3 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.

4.3.1 Detailed Description

A class which contains a various dispatch control logic. Intended to serve as a component class of Model.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Controller()

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

Constructor for the Controller class.

```
{
    //...

    return;
}   /* Controller() */
```

4.3.2.2 ∼Controller()

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

Destructor for the Controller class.

```
{
    //...

    return;
}   /* ~Controller() */
```

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

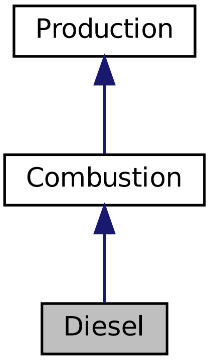
- header/Controller.h
- source/Controller.cpp

4.4 Diesel Class Reference

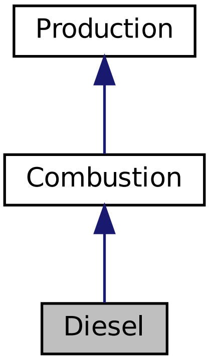
A derived class of the Combustion branch of Production which models production using a diesel generator.

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

Inheritance diagram for Diesel:



Collaboration diagram for Diesel:



Public Member Functions

- Diesel (void)

  Constructor (dummy) for the Diesel class.
- Diesel (int, DieselInputs)
- double requestProductionkW (int, double, double)

  Method which takes in production request, and then returns what the asset can deliver (subject to operating constraints, etc.).
- double commit (int, double, double, double)

  Method which takes in production and load for the current timestep, computes and records dispatch and curtailment, and then returns remaining load.
- ∼Diesel (void)

  Destructor for the Diesel class.

Public Attributes

- double fuel_cost_L

  The cost of fuel [1/L] (undefined currency).
- double minimum_load_ratio

  The minimum load ratio of the asset. That is, when the asset is producing, it must produce at least this ratio of its rated capacity.
- double minimum_runtime_hrs

  The minimum runtime [hrs] of the asset. This is the minimum time that must elapse between successive starts and stops.
- double time_since_last_start_hrs

  The time that has elapsed [hrs] since the last start of the asset.
- double CO2_emissions_intensity_kgL

  Carbon dioxide (CO2) emissions intensity [kg/L].
- double CO_emissions_intensity_kgL

  Carbon monoxide (CO) emissions intensity [kg/L].
- double NOx_emissions_intensity_kgL

Nitrogen oxide (NOx) emissions intensity [kg/L].

- double SOx_emissions_intensity_kgL

  Sulfur oxide (SOx) emissions intensity [kg/L].

- double CH4_emissions_intensity_kgL

  Methane (CH4) emissions intensity [kg/L].

- double PM_emissions_intensity_kgL

  Particulate Matter (PM) emissions intensity [kg/L].

4.4.1 Detailed Description

A derived class of the Combustion branch of Production which models production using a diesel generator.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Diesel() [1/2]

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

Constructor (dummy) for the Diesel class.

Constructor (intended) for the Diesel class.

Parameters

n_points	The number of points in the modelling time series.
diesel_inputs	A structure of Diesel constructor inputs.

```
{
    return;
}   /* Diesel() */
```

4.4.2.2 Diesel() [2/2]

```
Diesel::Diesel (
              int n_points,
              DieselInputs diesel_inputs )
                                                    :
Combustion(n_points, diesel_inputs.combustion_inputs)
{
    //  1. check inputs
    this->__checkInputs(diesel_inputs);

    //  2. set attributes
    this->fuel_cost_L = diesel_inputs.fuel_cost_L;

    this->minimum_load_ratio = diesel_inputs.minimum_load_ratio;
    this->minimum_runtime_hrs = diesel_inputs.minimum_runtime_hrs;
    this->time_since_last_start_hrs = 0;

    this->CO2_emissions_intensity_kgL = diesel_inputs.CO2_emissions_intensity_kgL;
    this->CO_emissions_intensity_kgL = diesel_inputs.CO_emissions_intensity_kgL;
    this->NOx_emissions_intensity_kgL = diesel_inputs.NOx_emissions_intensity_kgL;
    this->SOx_emissions_intensity_kgL = diesel_inputs.SOx_emissions_intensity_kgL;
    this->CH4_emissions_intensity_kgL = diesel_inputs.CH4_emissions_intensity_kgL;
    this->PM_emissions_intensity_kgL = diesel_inputs.PM_emissions_intensity_kgL;

    if (diesel_inputs.linear_fuel_slope_LkWh < 0) {
        this->linear_fuel_slope_LkWh = this->__getGenericFuelSlope();

    }

    if (diesel_inputs.linear_fuel_intercept_LkWh < 0) {
        this->linear_fuel_intercept_LkWh = this->__getGenericFuelIntercept();
    }

    if (diesel_inputs.capital_cost < 0) {
        this->capital_cost = this->__getGenericCapitalCost();
    }

    if (diesel_inputs.operation_maintenance_cost_kWh < 0) {
        this->operation_maintenance_cost_kWh = this->__getGenericOpMaintCost();
    }

    if (this->is_sunk) {
        this->capital_cost_vec[0] = this->capital_cost;
    }

    //  3. construction print
    if (this->print_flag) {
        std::cout « "Diesel object constructed at " « this « std::endl;
    }

    return;
}   /* Diesel() */
```

4.4.2.3 ∼Diesel()

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

Destructor for the Diesel class.

```
{
    //  1. destruction print
    if (this->print_flag) {
        std::cout « "Diesel object at " « this « " destroyed" « std::endl;
    }

    return;
}   /* ~Diesel() */
```

4.4.3 Member Function Documentation

4.4.3.1 commit()

```
double Diesel::commit (
               int timestep,
               double dt_hrs,
               double production_kW,
               double load_kW )  [virtual]
```

Method which takes in production and load for the current timestep, computes and records dispatch and curtailment, and then returns remaining load.

Parameters

timestep	The timestep (i.e., time series index) for the request.
dt_hrs	The interval of time [hrs] associated with the timestep.
production_kW	The production [kW] of the asset in this timestep.
load_kW	The load [kW] passed to the asset in this timestep.

Reimplemented from Combustion.

```
{
    //  1. handle start/stop, enforce minimum runtime constraint
    this->__handleStartStop(timestep, dt_hrs, production_kW);

    //  2. invoke base class method
    load_kW = Combustion :: commit(
        timestep,
        dt_hrs,
        production_kW,
        load_kW
    );

    if (this->is_running) {
        //  3. log time since last start
        this->time_since_last_start_hrs += dt_hrs;
    }

    return load_kW;
}   /* commit() */
```

4.4.3.2 requestProductionkW()

```
double Diesel::requestProductionkW (
            int timestep,
            double dt_hrs,
            double request_kW )  [virtual]
```

Method which takes in production request, and then returns what the asset can deliver (subject to operating constraints, etc.).

Parameters

timestep	The timestep (i.e., time series index) for the request.
dt_hrs	The interval of time [hrs] associated with the timestep.
request_kW	The requested production [kW].

Reimplemented from Combustion.

```
{
    double deliver_kW = request_kW;

    //  1. enforce capacity constraint
    if (deliver_kW > this->capacity_kW) {
        deliver_kW = this->capacity_kW;
    }

    //  2. enforce minimum load ratio
    if (deliver_kW < this->minimum_load_ratio * this->capacity_kW) {
        deliver_kW = this->minimum_load_ratio * this->capacity_kW;
    }

    return deliver_kW;
}   /* requestProductionkW() */
```

4.4.4 Member Data Documentation

4.4.4.1 CH4_emissions_intensity_kgL

```
double Diesel::CH4_emissions_intensity_kgL
```

Methane (CH4) emissions intensity [kg/L].

4.4.4.2 CO2_emissions_intensity_kgL

```
double Diesel::CO2_emissions_intensity_kgL
```

Carbon dioxide (CO2) emissions intensity [kg/L].

4.4.4.3 CO_emissions_intensity_kgL

```
double Diesel::CO_emissions_intensity_kgL
```

Carbon monoxide (CO) emissions intensity [kg/L].

4.4.4.4 fuel_cost_L

```
double Diesel::fuel_cost_L
```

The cost of fuel [1/L] (undefined currency).

4.4.4.5 minimum_load_ratio

```
double Diesel::minimum_load_ratio
```

The minimum load ratio of the asset. That is, when the asset is producing, it must produce at least this ratio of its rated capacity.

4.4.4.6 minimum_runtime_hrs

```
double Diesel::minimum_runtime_hrs
```

The minimum runtime [hrs] of the asset. This is the minimum time that must elapse between successive starts and stops.

4.4.4.7 NOx_emissions_intensity_kgL

```
double Diesel::NOx_emissions_intensity_kgL
```

Nitrogen oxide (NOx) emissions intensity [kg/L].

4.4.4.8 PM_emissions_intensity_kgL

```
double Diesel::PM_emissions_intensity_kgL
```

Particulate Matter (PM) emissions intensity [kg/L].

4.4.4.9 SOx_emissions_intensity_kgL

```
double Diesel::SOx_emissions_intensity_kgL
```

Sulfur oxide (SOx) emissions intensity [kg/L].

4.4.4.10 time_since_last_start_hrs

```
double Diesel::time_since_last_start_hrs
```

The time that has elapsed [hrs] since the last start of the asset.

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

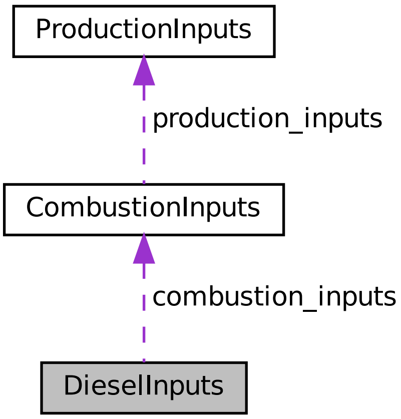
- header/Production/Combustion/Diesel.h
- source/Production/Combustion/Diesel.cpp

4.5 DieselInputs Struct Reference

A structure which bundles the necessary inputs for the Diesel constructor. Provides default values for every necessary input. Note that this structure encapsulates CombustionInputs.

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

Collaboration diagram for DieselInputs:



Public Attributes

- CombustionInputs combustion_inputs

  An encapsulated CombustionInputs instance.
- double capital_cost = -1

  The capital cost of the asset (undefined currency). -1 is a sentinel value, which triggers a generic cost model on construction (in fact, any negative value here will trigger). Note that the generic cost model is in terms of Canadian dollars [CAD].
- double operation_maintenance_cost_kWh = -1

  The operation and maintenance cost of the asset [1/kWh] (undefined currency). This is a cost incurred per unit of energy produced. -1 is a sentinel value, which triggers a generic cost model on construction (in fact, any negative value here will trigger). Note that the generic cost model is in terms of Canadian dollars [CAD/kWh].
- double fuel_cost_L = 1.70

  The cost of fuel [1/L] (undefined currency).
- double minimum_load_ratio = 0.2

  The minimum load ratio of the asset. That is, when the asset is producing, it must produce at least this ratio of its rated capacity.
- double minimum_runtime_hrs = 1

  The minimum runtime [hrs] of the asset. This is the minimum time that must elapse between successive starts and stops.
- double linear_fuel_slope_LkWh = -1

  The slope [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced. -1 is a sentinel value, which triggers a generic fuel consumption model on construction (in fact, any negative value here will trigger).

- double linear_fuel_intercept_LkWh = -1

  The intercept [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced. -1 is a sentinel value, which triggers a generic fuel consumption model on construction (in fact, any negative value here will trigger).
- double CO2_emissions_intensity_kgL = 2.7

  Carbon dioxide (CO2) emissions intensity [kg/L].
- double CO_emissions_intensity_kgL = 0.0178

  Carbon monoxide (CO) emissions intensity [kg/L].
- double NOx_emissions_intensity_kgL = 0.0014

  Nitrogen oxide (NOx) emissions intensity [kg/L].
- double SOx_emissions_intensity_kgL = 0.0042

  Sulfur oxide (SOx) emissions intensity [kg/L].
- double CH4_emissions_intensity_kgL = 0.0007

  Methane (CH4) emissions intensity [kg/L].
- double PM_emissions_intensity_kgL = 0.0001

  Particulate Matter (PM) emissions intensity [kg/L].

4.5.1 Detailed Description

A structure which bundles the necessary inputs for the Diesel constructor. Provides default values for every necessary input. Note that this structure encapsulates CombustionInputs.

4.5.2 Member Data Documentation

4.5.2.1 capital_cost

```
double DieselInputs::capital_cost = -1
```

The capital cost of the asset (undefined currency). -1 is a sentinel value, which triggers a generic cost model on construction (in fact, any negative value here will trigger). Note that the generic cost model is in terms of Canadian dollars [CAD].

4.5.2.2 CH4_emissions_intensity_kgL

```
double DieselInputs::CH4_emissions_intensity_kgL = 0.0007
```

Methane (CH4) emissions intensity [kg/L].

4.5.2.3 CO2_emissions_intensity_kgL

```
double DieselInputs::CO2_emissions_intensity_kgL = 2.7
```

Carbon dioxide (CO2) emissions intensity [kg/L].

4.5.2.4 CO_emissions_intensity_kgL

```
double DieselInputs::CO_emissions_intensity_kgL = 0.0178
```

Carbon monoxide (CO) emissions intensity [kg/L].

4.5.2.5 combustion_inputs

```
CombustionInputs DieselInputs::combustion_inputs
```

An encapsulated CombustionInputs instance.

4.5.2.6 fuel_cost_L

```
double DieselInputs::fuel_cost_L = 1.70
```

The cost of fuel [1/L] (undefined currency).

4.5.2.7 linear_fuel_intercept_LkWh

```
double DieselInputs::linear_fuel_intercept_LkWh = -1
```

The intercept [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced. -1 is a sentinel value, which triggers a generic fuel consumption model on construction (in fact, any negative value here will trigger).

4.5.2.8 linear_fuel_slope_LkWh

```
double DieselInputs::linear_fuel_slope_LkWh = -1
```

The slope [L/kWh] to use in computing linearized fuel consumption. This is fuel consumption per unit energy produced. -1 is a sentinel value, which triggers a generic fuel consumption model on construction (in fact, any negative value here will trigger).

4.5.2.9 minimum_load_ratio

```
double DieselInputs::minimum_load_ratio = 0.2
```

The minimum load ratio of the asset. That is, when the asset is producing, it must produce at least this ratio of its rated capacity.

4.5.2.10 minimum_runtime_hrs

```
double DieselInputs::minimum_runtime_hrs = 1
```

The minimum runtime [hrs] of the asset. This is the minimum time that must elapse between successive starts and stops.

4.5.2.11 NOx_emissions_intensity_kgL

```
double DieselInputs::NOx_emissions_intensity_kgL = 0.0014
```

Nitrogen oxide (NOx) emissions intensity [kg/L].

4.5.2.12 operation_maintenance_cost_kWh

```
double DieselInputs::operation_maintenance_cost_kWh = -1
```

The operation and maintenance cost of the asset [1/kWh] (undefined currency). This is a cost incurred per unit of energy produced. -1 is a sentinel value, which triggers a generic cost model on construction (in fact, any negative value here will trigger). Note that the generic cost model is in terms of Canadian dollars [CAD/kWh].

4.5.2.13 PM_emissions_intensity_kgL

```
double DieselInputs::PM_emissions_intensity_kgL = 0.0001
```

Particulate Matter (PM) emissions intensity [kg/L].

4.5.2.14 SOx_emissions_intensity_kgL

```
double DieselInputs::SOx_emissions_intensity_kgL = 0.0042
```

Sulfur oxide (SOx) emissions intensity [kg/L].

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

- header/Production/Combustion/Diesel.h

4.6 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.

4.6.1 Detailed Description

A class which contains time and electrical load data. Intended to serve as a component class of Model.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ElectricalLoad()

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

Constructor for the ElectricalLoad class.

```
{
    //...

    return;
}   /* ElectricalLoad() */
```

4.6.2.2 ∼ElectricalLoad()

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

Destructor for the ElectricalLoad class.

```
{
    //...

    return;
}   /* ~ElectricalLoad() */
```

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

- header/ElectricalLoad.h
- source/ElectricalLoad.cpp

4.7 Emissions Struct Reference

A structure which bundles the emitted masses of various emissions chemistries.

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

Public Attributes

- double CO2_kg = 0

  The mass of carbon dioxide (CO2) emitted [kg].
- double CO_kg = 0

  The mass of carbon monoxide (CO) emitted [kg].
- double NOx_kg = 0

  The mass of nitrogen oxides (NOx) emitted [kg].
- double SOx_kg = 0

  The mass of sulfur oxides (SOx) emitted [kg].
- double CH4_kg = 0

  The mass of methane (CH4) emitted [kg].
- double PM_kg = 0

  The mass of particulate matter (PM) emitted [kg].

4.7.1 Detailed Description

A structure which bundles the emitted masses of various emissions chemistries.

4.7.2 Member Data Documentation

4.7.2.1 CH4_kg

```
double Emissions::CH4_kg = 0
```

The mass of methane (CH4) emitted [kg].

4.7.2.2 CO2_kg

```
double Emissions::CO2_kg = 0
```

The mass of carbon dioxide (CO2) emitted [kg].

4.7.2.3 CO_kg

```
double Emissions::CO_kg = 0
```

The mass of carbon monoxide (CO) emitted [kg].

4.7.2.4 NOx_kg

```
double Emissions::NOx_kg = 0
```

The mass of nitrogen oxides (NOx) emitted [kg].

4.7.2.5 PM_kg

```
double Emissions::PM_kg = 0
```

The mass of particulate matter (PM) emitted [kg].

4.7.2.6 SOx_kg

```
double Emissions::SOx_kg = 0
```

The mass of sulfur oxides (SOx) emitted [kg].

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

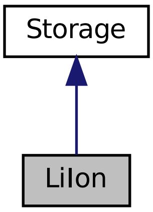
- header/Production/Combustion/Combustion.h

4.8 LiIon Class Reference

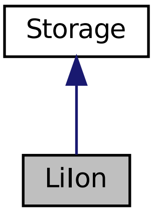
A derived class of Storage which models energy storage by way of lithium-ion batteries.

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

Inheritance diagram for LiIon:



Collaboration diagram for LiIon:



Public Member Functions

- LiIon (void)

    Constructor for the LiIon class.
- ∼LiIon (void)

    Destructor for the LiIon class.

4.8.1 Detailed Description

A derived class of Storage which models energy storage by way of lithium-ion batteries.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 LiIon()

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

Constructor for the LiIon class.

```
                   :
Storage()
{
    //...

    return;
}   /* LiIon() */
```

4.8.2.2 ∼LiIon()

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

Destructor for the LiIon class.

```
{
    //...

    return;
}   /* ~LiIon() */
```

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

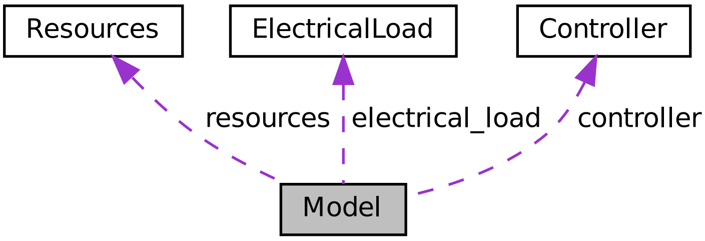
- header/Storage/LiIon.h
- source/Storage/LiIon.cpp

4.9 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.

4.9.1 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.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Model()

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

Constructor for the Model class.

```
{
    //...

    return;
}   /* Model() */
```

4.9.2.2 ∼Model()

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

Destructor for the Model class.

```
{
    //...

    return;
}   /* ~Model() */
```

4.9.3 Member Data Documentation

4.9.3.1 combustion_ptr_vec

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

A vector of pointers to the various Combustion assets in the Model.

4.9.3.2 controller

```
Controller Model::controller
```

Controller component of Model.

4.9.3.3 electrical_load

```
ElectricalLoad Model::electrical_load
```

ElectricalLoad component of Model.

4.9.3.4 renewable_ptr_vec

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

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

4.9.3.5 resources

```
Resources Model::resources
```

Resources component of Model.

4.9.3.6 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:

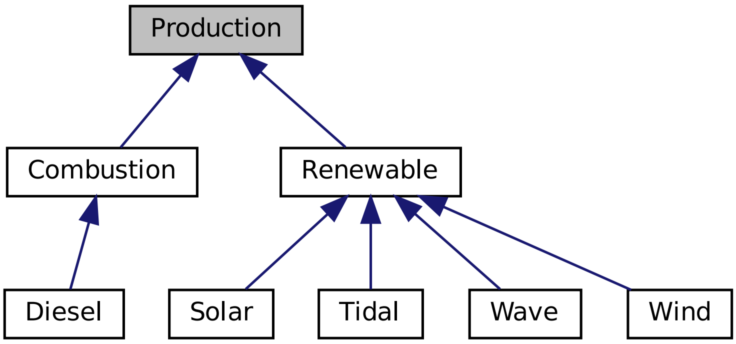
- header/Model.h
- source/Model.cpp

4.10 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 (dummy) for the Production class.
- Production (int, ProductionInputs)

  Constructor (intended) for the Production class.
- virtual double commit (int, double, double, double)

  Method which takes in production and load for the current timestep, computes and records dispatch and curtailment, and then returns remaining load.
- virtual ∼Production (void)

  Destructor for the Production class.

Public Attributes

- bool print_flag

  A flag which indicates whether or not object construct/destruction should be verbose.
- bool is_running

  A boolean which indicates whether or not the asset is running.
- bool is_sunk

  A boolean which indicates whether or not the asset should be considered a sunk cost (i.e., capital cost incurred at the start of the model, or no).
- int n_points

  The number of points in the modelling time series.
- int n_starts

  The number of times the asset has been started.
- double running_hours

  The number of hours for which the assset has been operating.
- double capacity_kW

  The rated production capacity [kW] of the asset.
- double real_discount_annual

  The real, annual discount rate used in computing model economics. Is computed from the given nominal inflation and discount rates.
- double capital_cost

  The capital cost of the asset (undefined currency).
- double operation_maintenance_cost_kWh

  The operation and maintenance cost of the asset [1/kWh] (undefined currency). This is a cost incurred per unit of energy produced.
- double net_present_cost

  The net present cost of this asset.
- double levellized_cost_of_energy_kWh

  The levellized cost of energy [1/kWh] (undefined currency) of this asset. This metric considers only dispatched and stored energy.
- std::vector< bool > is_running_vec

  A boolean vector for tracking if the asset is running at a particular point in time.
- std::vector< double > production_vec_kW

  A vector of production [kW] at each point in the modelling time series.
- std::vector< double > dispatch_vec_kW

  A vector of dispatch [kW] at each point in the modelling time series. Dispatch is the amount of production that is sent to the grid to satisfy load.
- std::vector< double > storage_vec_kW

  A vector of storage [kW] at each point in the modelling time series. Storage is the amount of production that is sent to storage.
- std::vector< double > curtailment_vec_kW

  A vector of curtailment [kW] at each point in the modelling time series. Curtailment is the amount of production that can be neither dispatched nor stored, and is hence curtailed.
- std::vector< double > capital_cost_vec

  A vector of capital costs (undefined currency) incurred over each modelling time step. These costs are not discounted (i.e., these are nominal costs).
- std::vector< double > operation_maintenance_cost_vec

  A vector of operation and maintenance costs (undefined currency) incurred over each modelling time step. These costs are not discounted (i.e., these are nominal costs).

4.10.1 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.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Production() [1/2]

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

Constructor (dummy) for the Production class.

```
{
    return;
}   /* Production() */
```

4.10.2.2 Production() [2/2]

```
Production::Production (
             int n_points,
             ProductionInputs production_inputs )
```

Constructor (intended) for the Production class.

Parameters

n_points	The number of points in the modelling time series.
production_inputs	A structure of Production constructor inputs.

```
{
    //  1. check inputs
    this->__checkInputs(n_points, production_inputs);

    //  2. set attributes
    this->print_flag = production_inputs.print_flag;
    this->is_running = false;

    this->n_points = n_points;
    this->n_starts = 0;

    this->running_hours = 0;

    this->capacity_kW = production_inputs.capacity_kW;

    this->real_discount_annual = this->__computeRealDiscountAnnual(
        production_inputs.nominal_inflation_annual,
        production_inputs.nominal_discount_annual
    );
    this->capital_cost = 0;
    this->operation_maintenance_cost_kWh = 0;
    this->net_present_cost = 0;
    this->levellized_cost_of_energy_kWh = 0;

    this->production_vec_kW.resize(this->n_points, 0);
    this->dispatch_vec_kW.resize(this->n_points, 0);
    this->storage_vec_kW.resize(this->n_points, 0);
    this->curtailment_vec_kW.resize(this->n_points, 0);

    this->capital_cost_vec.resize(this->n_points, 0);
    this->operation_maintenance_cost_vec.resize(this->n_points, 0);

    //  3. construction print
    if (this->print_flag) {
        std::cout « "Production object constructed at " « this « std::endl;
    }

    return;
}   /* Production() */
```

4.10.2.3 ∼Production()

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

Destructor for the Production class.

```
{
    //  1. destruction print
    if (this->print_flag) {
        std::cout « "Production object at " « this « " destroyed" « std::endl;
    }

    return;
}   /* ~Production() */
```

4.10.3 Member Function Documentation

4.10.3.1 commit()

```
double Production::commit (
              int timestep,
              double dt_hrs,
              double production_kW,
              double load_kW )  [virtual]
```

Method which takes in production and load for the current timestep, computes and records dispatch and curtailment, and then returns remaining load.

Parameters

timestep	The timestep (i.e., time series index) for the request.
dt_hrs	The interval of time [hrs] associated with the timestep.
production_kW	The production [kW] of the asset in this timestep.
load_kW	The load [kW] passed to the asset in this timestep.

Returns

load_kW The load [kW] remaining after the dispatch is deducted from it.

Reimplemented in Diesel, and Combustion.

```
{
    //  1. record production
    this->production_vec_kW[timestep] = production_kW;

    //  2. compute and record dispatch and curtailment
    double dispatch_kW = 0;
    double curtailment_kW = 0;

    if (production_kW > load_kW) {
        dispatch_kW = load_kW;
        curtailment_kW = production_kW - dispatch_kW;
    }

    else {
        dispatch_kW = production_kW;
    }

    this->dispatch_vec_kW[timestep] = dispatch_kW;
    this->curtailment_vec_kW[timestep] = curtailment_kW;

    //  3. update load
    load_kW -= dispatch_kW;

    if (this->is_running) {
        //  4. log running state, running hours
        this->is_running_vec[timestep] = this->is_running;
        this->running_hours += dt_hrs;

        //  5. incur capital and operating costs
        //...
    }


    return load_kW;
}
```

4.10.4 Member Data Documentation

4.10.4.1 capacity_kW

`double Production::capacity_kW`

The rated production capacity [kW] of the asset.

4.10.4.2 capital_cost

`double Production::capital_cost`

The capital cost of the asset (undefined currency).

4.10.4.3 capital_cost_vec

`std::vector<double> Production::capital_cost_vec`

A vector of capital costs (undefined currency) incurred over each modelling time step. These costs are not discounted (i.e., these are nominal costs).

4.10.4.4 curtailment_vec_kW

```
std::vector<double> Production::curtailment_vec_kW
```

A vector of curtailment [kW] at each point in the modelling time series. Curtailment is the amount of production that can be neither dispatched nor stored, and is hence curtailed.

4.10.4.5 dispatch_vec_kW

```
std::vector<double> Production::dispatch_vec_kW
```

A vector of dispatch [kW] at each point in the modelling time series. Dispatch is the amount of production that is sent to the grid to satisfy load.

4.10.4.6 is_running

```
bool Production::is_running
```

A boolean which indicates whether or not the asset is running.

4.10.4.7 is_running_vec

```
std::vector<bool> Production::is_running_vec
```

A boolean vector for tracking if the asset is running at a particular point in time.

4.10.4.8 is_sunk

```
bool Production::is_sunk
```

A boolean which indicates whether or not the asset should be considered a sunk cost (i.e., capital cost incurred at the start of the model, or no).

4.10.4.9 levellized_cost_of_energy_kWh

```
double Production::levellized_cost_of_energy_kWh
```

The levellized cost of energy [1/kWh] (undefined currency) of this asset. This metric considers only dispatched and stored energy.

4.10.4.10 n_points

```
int Production::n_points
```

The number of points in the modelling time series.

4.10.4.11 n_starts

```
int Production::n_starts
```

The number of times the asset has been started.

4.10.4.12 net_present_cost

```
double Production::net_present_cost
```

The net present cost of this asset.

4.10.4.13 operation_maintenance_cost_kWh

```
double Production::operation_maintenance_cost_kWh
```

The operation and maintenance cost of the asset [1/kWh] (undefined currency). This is a cost incurred per unit of energy produced.

4.10.4.14 operation_maintenance_cost_vec

```
std::vector<double> Production::operation_maintenance_cost_vec
```

A vector of operation and maintenance costs (undefined currency) incurred over each modelling time step. These costs are not discounted (i.e., these are nominal costs).

4.10.4.15 print_flag

```
bool Production::print_flag
```

A flag which indicates whether or not object construct/destruction should be verbose.

4.10.4.16 production_vec_kW

```
std::vector<double> Production::production_vec_kW
```

A vector of production [kW] at each point in the modelling time series.

4.10.4.17 real_discount_annual

```
double Production::real_discount_annual
```

The real, annual discount rate used in computing model economics. Is computed from the given nominal inflation and discount rates.

4.10.4.18 running_hours

```
double Production::running_hours
```

The number of hours for which the assset has been operating.

4.10.4.19 storage_vec_kW

```
std::vector<double> Production::storage_vec_kW
```

A vector of storage [kW] at each point in the modelling time series. Storage is the amount of production that is sent to storage.

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

- header/Production/Production.h
- source/Production/Production.cpp

4.11 ProductionInputs Struct Reference

A structure which bundles the necessary inputs for the Production constructor. Provides default values for every necessary input.

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

Public Attributes

- bool print_flag = false

  A flag which indicates whether or not object construct/destruction should be verbose.
- bool is_sunk = false

  A boolean which indicates whether or not the asset should be considered a sunk cost (i.e., capital cost incurred at the start of the model, or no).
- double capacity_kW = 100

  The rated production capacity [kW] of the asset.
- double nominal_inflation_annual = 0.02

  The nominal, annual inflation rate to use in computing model economics.
- double nominal_discount_annual = 0.04

  The nominal, annual discount rate to use in computing model economics.

4.11.1 Detailed Description

A structure which bundles the necessary inputs for the Production constructor. Provides default values for every necessary input.

4.11.2 Member Data Documentation

4.11.2.1 capacity_kW

```
double ProductionInputs::capacity_kW = 100
```

The rated production capacity [kW] of the asset.

4.11.2.2 is_sunk

```
bool ProductionInputs::is_sunk = false
```

A boolean which indicates whether or not the asset should be considered a sunk cost (i.e., capital cost incurred at the start of the model, or no).

4.11.2.3 nominal_discount_annual

```
double ProductionInputs::nominal_discount_annual = 0.04
```

The nominal, annual discount rate to use in computing model economics.

4.11.2.4 nominal_inflation_annual

```
double ProductionInputs::nominal_inflation_annual = 0.02
```

The nominal, annual inflation rate to use in computing model economics.

4.11.2.5 print_flag

```
bool ProductionInputs::print_flag = false
```

A flag which indicates whether or not object construct/destruction should be verbose.

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

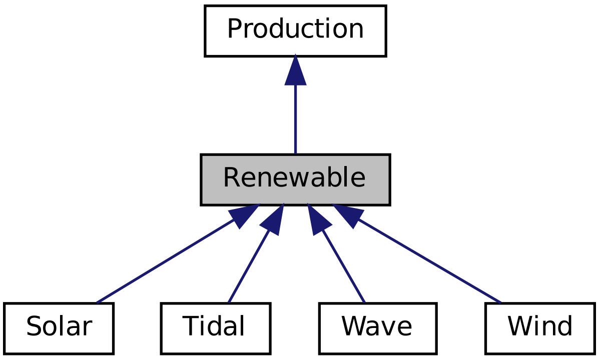
- header/Production/Production.h

4.12 Renewable Class Reference

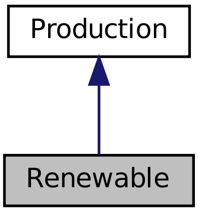
The root of the Renewable branch of the Production hierarchy. This branch contains derived classes which model the renewable production of energy.

```
#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.

Additional Inherited Members

4.12.1 Detailed Description

The root of the Renewable branch of the Production hierarchy. This branch contains derived classes which model the renewable production of energy.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 Renewable()

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

Constructor for the Renewable class.

```
                           :
Production()
{
    //...

    return;
}   /* Renewable() */
```

4.12.2.2 ∼Renewable()

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

Destructor for the Renewable class.

```
{
    //...

    return;
}   /* ~Renewable() */
```

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

- header/Production/Renewable/Renewable.h
- source/Production/Renewable/Renewable.cpp

4.13 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.

4.13.1 Detailed Description

A class which contains renewable resource data. Intended to serve as a component class of Model.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 Resources()

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

Constructor for the Resources class.

```
{
    //...

    return;
}   /* Resources() */
```

4.13.2.2 ∼Resources()

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

Destructor for the Resources class.

```
{
    //...

    return;
}   /* ~Resources() */
```

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

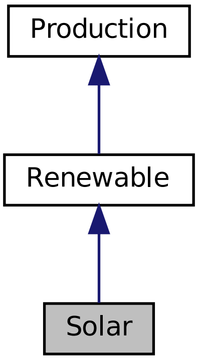
- header/Resources.h
- source/Resources.cpp

4.14 Solar Class Reference

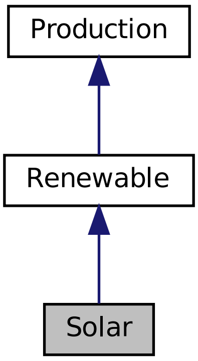
A derived class of the Renewable branch of Production which models solar production.

```
#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.

Additional Inherited Members

4.14.1 Detailed Description

A derived class of the Renewable branch of Production which models solar production.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 Solar()

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

Constructor for the Solar class.

```
                    :
Renewable()
{
    //...

    return;
}   /* Solar() */
```

4.14.2.2 ∼Solar()

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

Destructor for the Solar class.

```
{
    //...

    return;
}   /* ~Solar() */
```

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

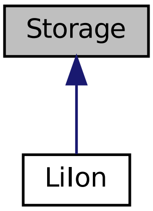
- header/Production/Renewable/Solar.h
- source/Production/Renewable/Solar.cpp

4.15 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.

4.15.1 Detailed Description

The base class of the Storage hierarchy. This hierarchy contains derived classes which model the storage of energy.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Storage()

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

Constructor for the Storage class.

```
{
    //...

    return;
}   /* Storage() */
```

4.15.2.2 ∼Storage()

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

Destructor for the Storage class.

```
{
    //...

    return;
}   /* ~Storage() */
```

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

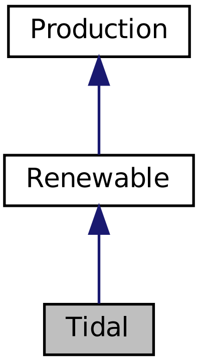
- header/Storage/Storage.h
- source/Storage/Storage.cpp

4.16 Tidal Class Reference

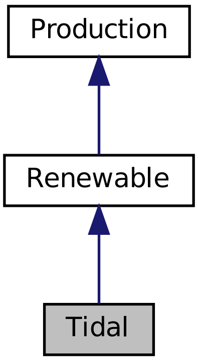
A derived class of the Renewable branch of Production which models tidal production.

```
#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.

Additional Inherited Members

4.16.1 Detailed Description

A derived class of the Renewable branch of Production which models tidal production.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Tidal()

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

Constructor for the Tidal class.

```
                   :
Renewable()
{
    //...

    return;
}   /* Tidal() */
```

4.16.2.2 ∼Tidal()

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

Destructor for the Tidal class.

```
{
    //...

    return;
}   /* ~Tidal() */
```

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

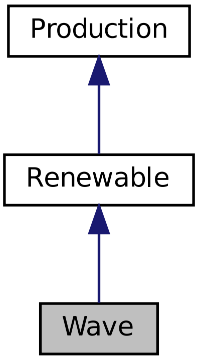
- header/Production/Renewable/Tidal.h
- source/Production/Renewable/Tidal.cpp

4.17 Wave Class Reference

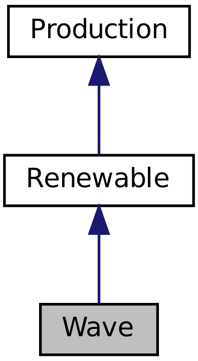
A derived class of the Renewable branch of Production which models wave production.

```
#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.

Additional Inherited Members

4.17.1 Detailed Description

A derived class of the Renewable branch of Production which models wave production.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Wave()

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

Constructor for the Wave class.

```
                :
Renewable()
{
    //...

    return;
}   /* Wave() */
```

4.17.2.2 ∼Wave()

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

Destructor for the Wave class.

```
{
    //...

    return;
}   /* ~Wave() */
```

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

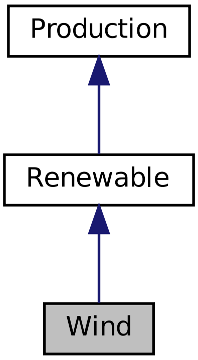
- header/Production/Renewable/Wave.h
- source/Production/Renewable/Wave.cpp

4.18 Wind Class Reference

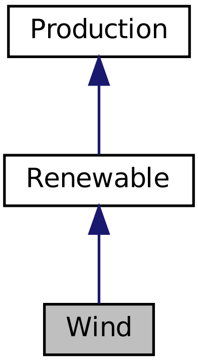
A derived class of the Renewable branch of Production which models wind production.

`#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.

Additional Inherited Members

4.18.1 Detailed Description

A derived class of the Renewable branch of Production which models wind production.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Wind()

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

Constructor for the Wind class.

```
                    :
Renewable()
{
    //...

    return;
}   /* Wind() */
```

4.18.2.2 ∼Wind()

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

Destructor for the Wind class.

```
{
    //...

    return;
}   /* ~Wind() */
```

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

- header/Production/Renewable/Wind.h
- source/Production/Renewable/Wind.cpp

Chapter 5

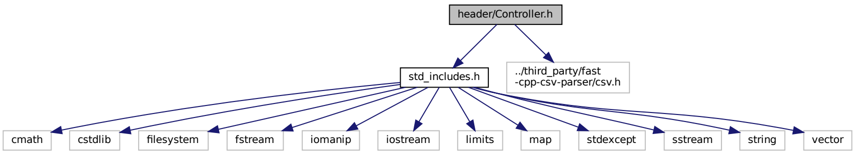
File Documentation

5.1 header/Controller.h File Reference

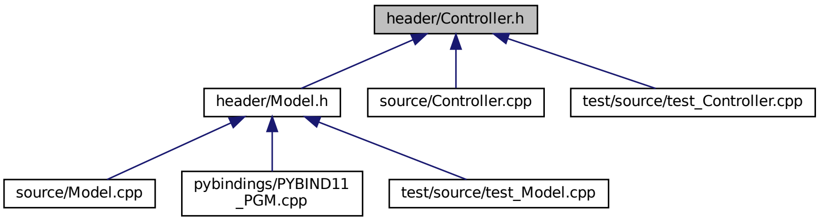
Header file the Controller class.

```
#include "std_includes.h"
#include "../third_party/fast-cpp-csv-parser/csv.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

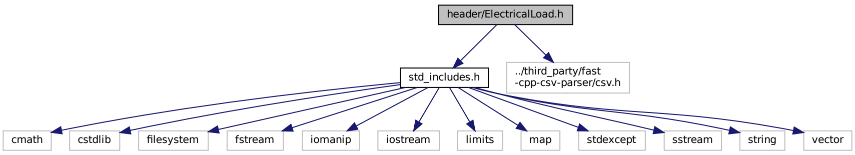
Header file the Controller class.

5.2 header/ElectricalLoad.h File Reference

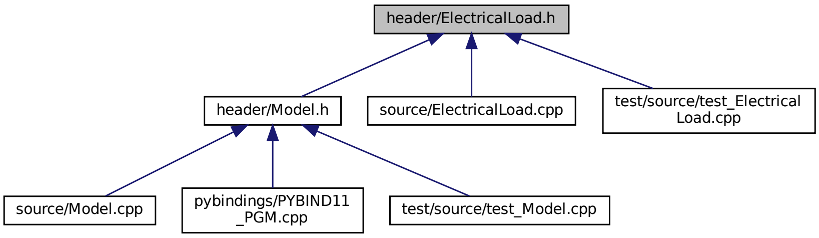
Header file the ElectricalLoad class.

```
#include "std_includes.h"
#include "../third_party/fast-cpp-csv-parser/csv.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

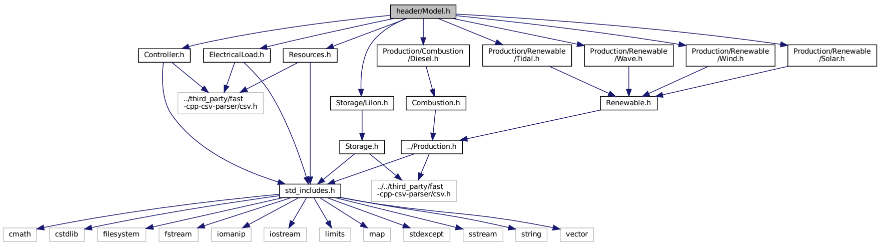
Header file the ElectricalLoad class.

5.3 header/Model.h File Reference

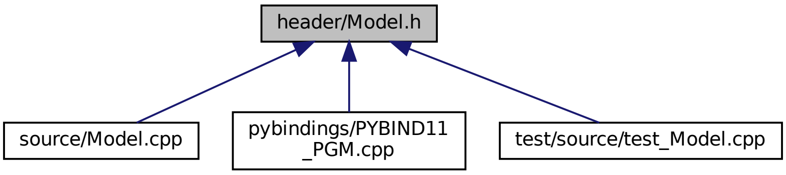
Header file the Model class.

```
#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:


header/Model.h
Controller.h
ElectricalLoad.h
Resources.h
Production/Combustion /Diesel.h
Production/Renewable /Tidal.h
Production/Renewable /Wave.h
Production/Renewable /Wind.h
Production/Renewable /Solar.h
../third_party/fast -cpp-csv-parser/csv.h
Storage/LiIon.h
Combustion.h
Renewable.h
Storage.h
../Production.h
std_includes.h
../../third_party/fast -cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

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

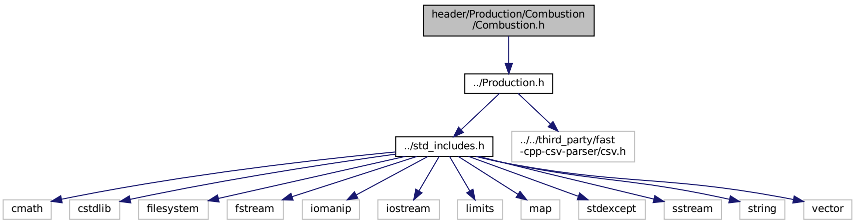
Header file the Model class.

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

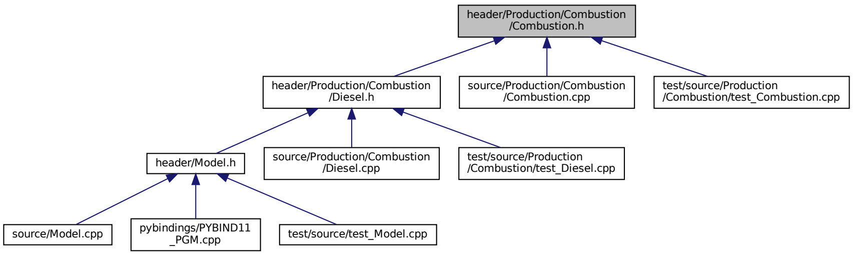
Header file the Combustion class.

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

Include dependency graph for Combustion.h:



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


header/Production/Combustion
/Combustion.h
header/Production/Combustion
/Diesel.h
source/Production/Combustion
/Combustion.cpp
test/source/Production
/Combustion/test_Combustion.cpp
header/Model.h
source/Production/Combustion
/Diesel.cpp
test/source/Production
/Combustion/test_Diesel.cpp
source/Model.cpp
pybindings/PYBIND11
_PGM.cpp
test/source/test_Model.cpp

Classes

- struct CombustionInputs

  A structure which bundles the necessary inputs for the Combustion constructor. Provides default values for every necessary input. Note that this structure encapsulates ProductionInputs.
- struct Emissions

  A structure which bundles the emitted masses of various emissions chemistries.
- 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.

Enumerations

- enum CombustionType { DIESEL , N_COMBUSTION_TYPES }

  An enumeration of the types of Combustion asset supported by PGMcpp.

5.4.1 Detailed Description

Header file the Combustion class.

5.4.2 Enumeration Type Documentation

5.4.2.1 CombustionType

`enum CombustionType`

An enumeration of the types of Combustion asset supported by PGMcpp.

Enumerator

DIESEL	A diesel generator.
N_COMBUSTION_TYPES	A simple hack to get the number of elements in CombustionType.

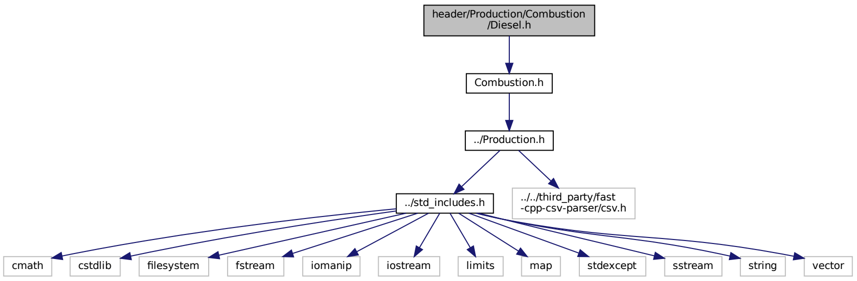
```
                    {
    DIESEL,
    N_COMBUSTION_TYPES
};
```

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

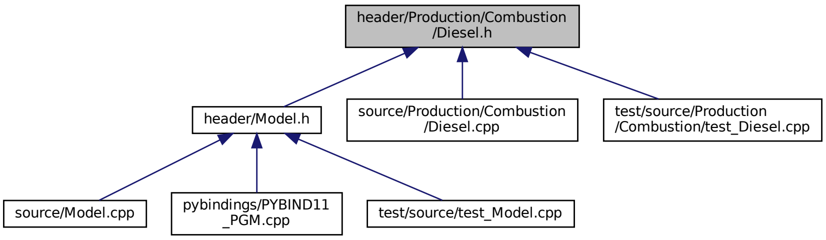
Header file the Diesel class.

`#include "Combustion.h"`

Include dependency graph for Diesel.h:



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


header/Production/Combustion
/Diesel.h
header/Model.h
source/Production/Combustion
/Diesel.cpp
test/source/Production
/Combustion/test_Diesel.cpp
source/Model.cpp
pybindings/PYBIND11
_PGM.cpp
test/source/test_Model.cpp

Classes

- struct DieselInputs

  A structure which bundles the necessary inputs for the Diesel constructor. Provides default values for every necessary input. Note that this structure encapsulates CombustionInputs.

- class Diesel

  A derived class of the Combustion branch of Production which models production using a diesel generator.

5.5.1 Detailed Description

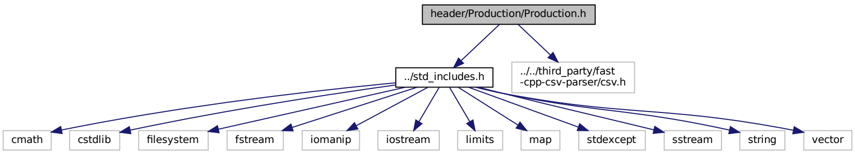
Header file the Diesel class.

5.6 header/Production/Production.h File Reference

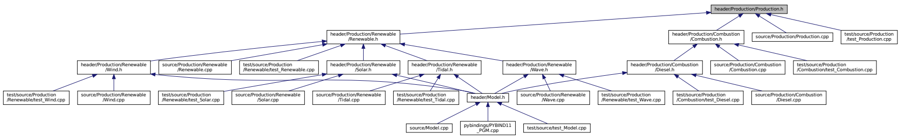
Header file the Production class.

```
#include "../std_includes.h"
#include "../../third_party/fast-cpp-csv-parser/csv.h"
```

Include dependency graph for Production.h:



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


header/Production/Production.h
header/Production/Renewable
/Renewable.h
header/Production/Combustion
/Combustion.h
source/Production/Production.cpp
test/source/Production
/test_Production.cpp
header/Production/Renewable
/Wind.h
source/Production/Renewable
/Renewable.cpp
test/source/Production
/Renewable/test_Renewable.cpp
header/Production/Renewable
/Solar.h
header/Production/Renewable
/Tidal.h
header/Production/Renewable
/Wave.h
header/Production/Combustion
/Diesel.h
source/Production/Combustion
/Combustion.cpp
test/source/Production
/Combustion/test_Combustion.cpp
test/source/Production
/Renewable/test_Wind.cpp
source/Production/Renewable
/Wind.cpp
test/source/Production
/Renewable/test_Solar.cpp
source/Production/Renewable
/Solar.cpp
source/Production/Renewable
/Tidal.cpp
test/source/Production
/Renewable/test_Tidal.cpp
header/Model.h
source/Production/Renewable
/Wave.cpp
test/source/Production
/Renewable/test_Wave.cpp
test/source/Production
/Combustion/test_Diesel.cpp
source/Production/Combustion
/Diesel.cpp
source/Model.cpp
pybindings/PYBIND11
_PGM.cpp
test/source/test_Model.cpp

Classes

- struct ProductionInputs

  A structure which bundles the necessary inputs for the Production constructor. Provides default values for every necessary input.

- 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

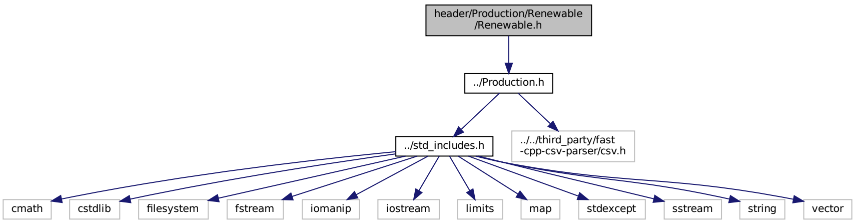
Header file the Production class.

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

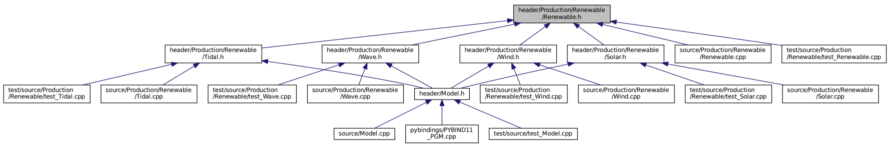
Header file the Renewable class.

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

Include dependency graph for Renewable.h:



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


header/Production/Renewable
/Renewable.h
header/Production/Renewable
/Tidal.h
header/Production/Renewable
/Wave.h
header/Production/Renewable
/Wind.h
header/Production/Renewable
/Solar.h
source/Production/Renewable
/Renewable.cpp
test/source/Production
/Renewable/test_Renewable.cpp
test/source/Production
/Renewable/test_Tidal.cpp
source/Production/Renewable
/Tidal.cpp
test/source/Production
/Renewable/test_Wave.cpp
source/Production/Renewable
/Wave.cpp
header/Model.h
test/source/Production
/Renewable/test_Wind.cpp
source/Production/Renewable
/Wind.cpp
test/source/Production
/Renewable/test_Solar.cpp
source/Production/Renewable
/Solar.cpp
source/Model.cpp
pybindings/PYBIND11
_PGM.cpp
test/source/test_Model.cpp

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

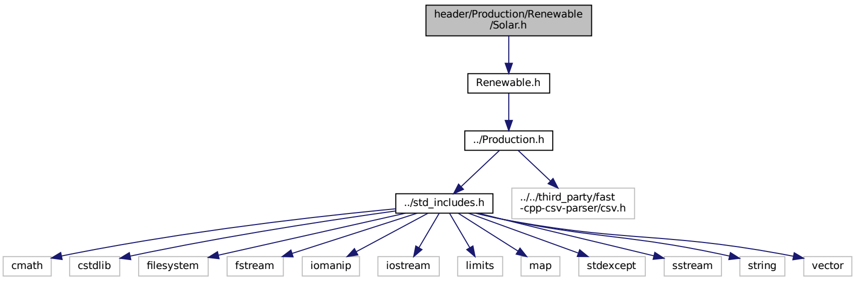
Header file the Renewable class.

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

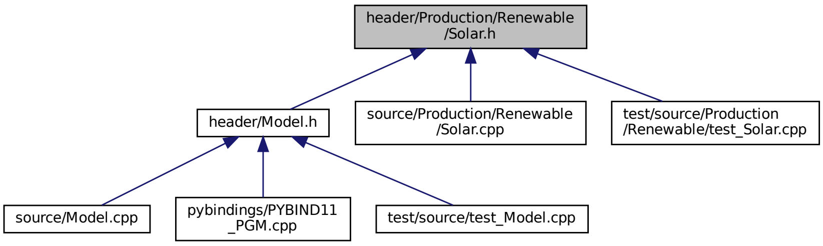
Header file the Solar class.

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

Include dependency graph for Solar.h:



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



header/Production/Renewable/Solar.h
header/Model.h
source/Production/Renewable/Solar.cpp
test/source/Production/Renewable/test_Solar.cpp
source/Model.cpp
pybindings/PYBIND11_PGM.cpp
test/source/test_Model.cpp

Classes

- class Solar

  A derived class of the Renewable branch of Production which models solar production.

5.8.1 Detailed Description

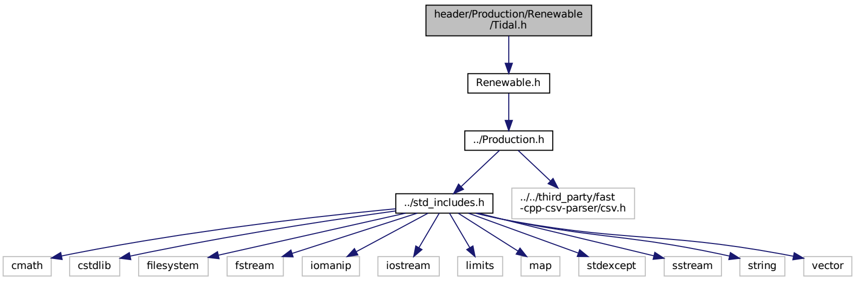
Header file the Solar class.

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

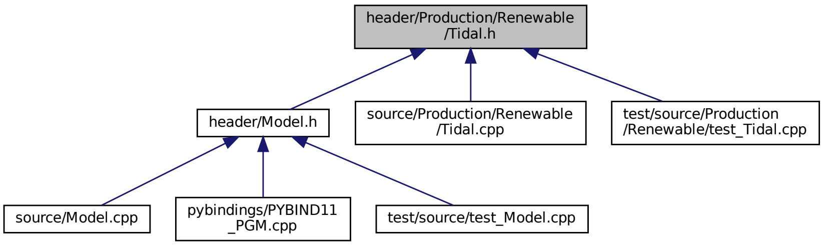
Header file the Tidal class.

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

Include dependency graph for Tidal.h:



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


header/Production/Renewable
/Tidal.h
header/Model.h
source/Production/Renewable
/Tidal.cpp
test/source/Production
/Renewable/test_Tidal.cpp
source/Model.cpp
pybindings/PYBIND11
_PGM.cpp
test/source/test_Model.cpp

Classes

- class Tidal

  A derived class of the Renewable branch of Production which models tidal production.

5.9.1 Detailed Description

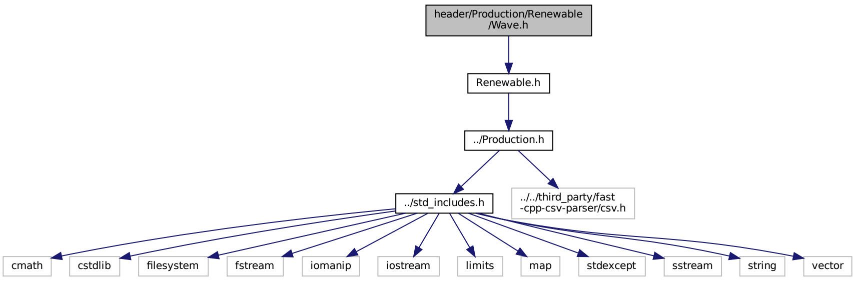
Header file the Tidal class.

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

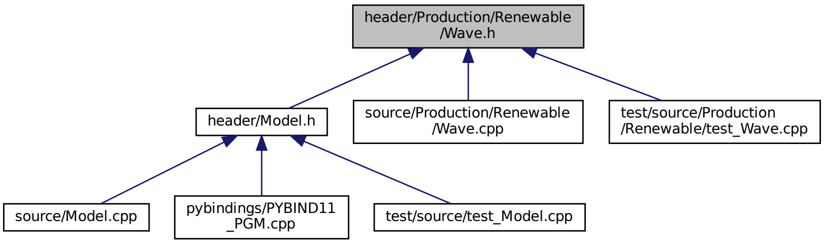
Header file the Wave class.

```
#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

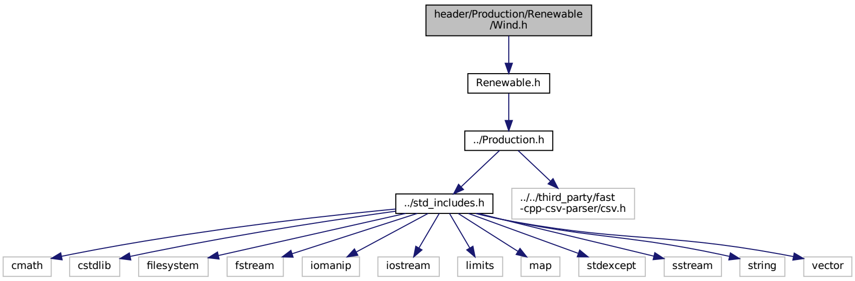
Header file the Wave class.

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

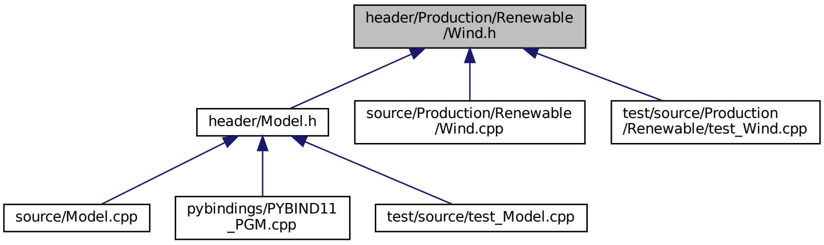
Header file the Wind class.

`#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

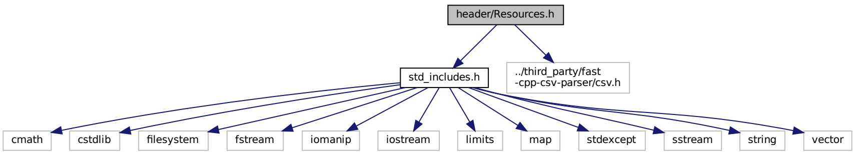
Header file the Wind class.

5.12 header/Resources.h File Reference

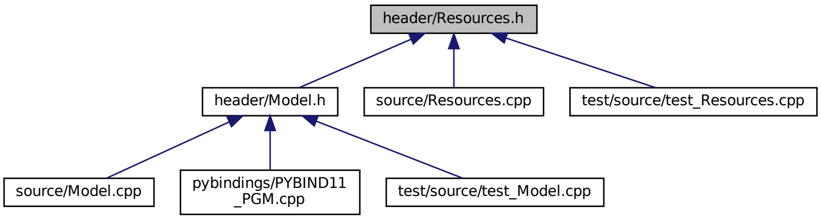
Header file the Resources class.

```
#include "std_includes.h"
#include "../third_party/fast-cpp-csv-parser/csv.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

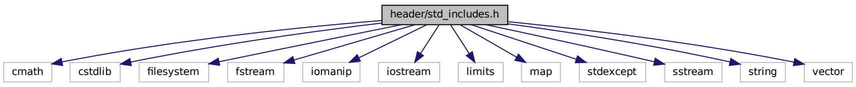
Header file the Resources class.

5.13 header/std_includes.h File Reference

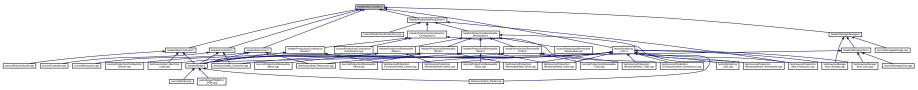
Header file which simply batches together the usual, standard includes.

```
#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

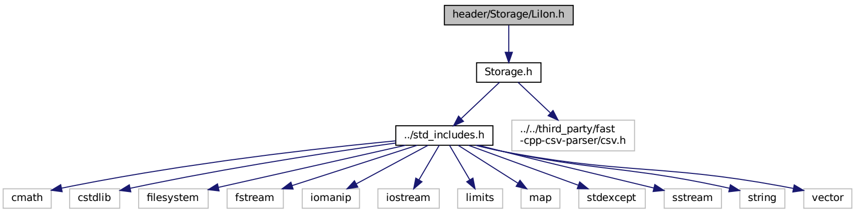
Header file which simply batches together the usual, standard includes.

5.14 header/Storage/LiIon.h File Reference

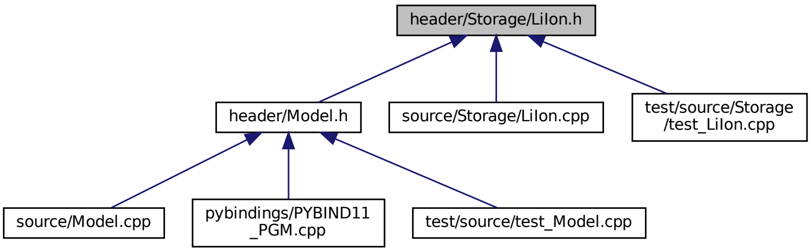
Header file the LiIon class.

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

Include dependency graph for LiIon.h:



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



Classes

- class LiIon

    A derived class of Storage which models energy storage by way of lithium-ion batteries.

5.14.1 Detailed Description

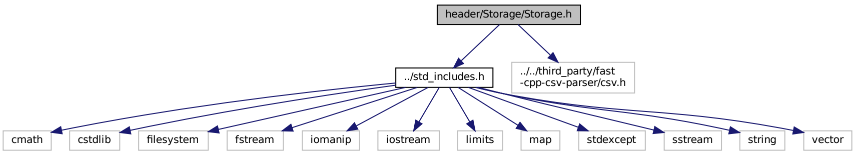
Header file the LiIon class.

5.15 header/Storage/Storage.h File Reference

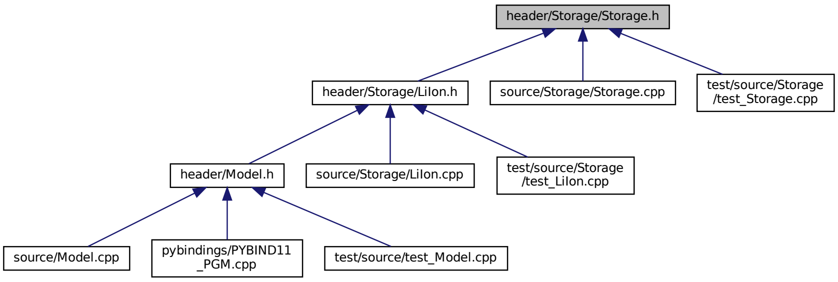
Header file the Storage class.

```
#include "../std_includes.h"
#include "../../third_party/fast-cpp-csv-parser/csv.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.

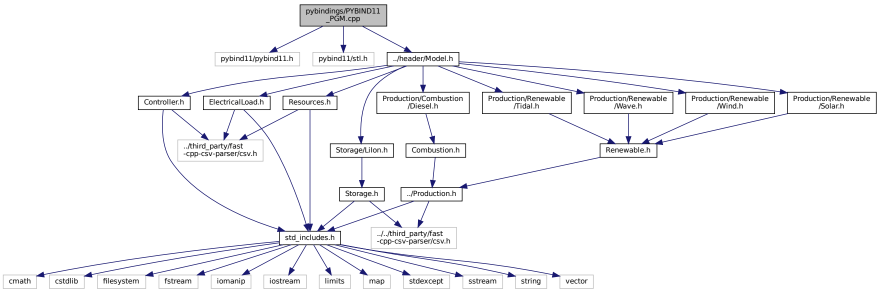
5.16 pybindings/PYBIND11_PGM.cpp File Reference

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 )
```

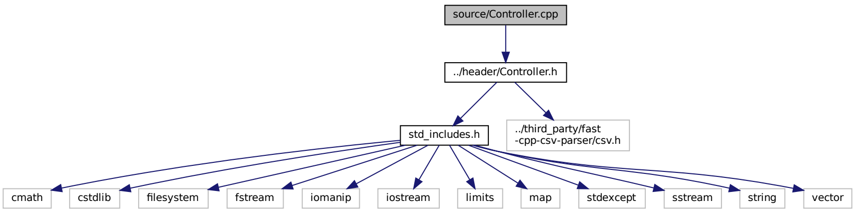
```
{

// ================ Controller ================ //
/*
pybind11::class_<Controller>(m, "Controller")
    .def(pybind11::init());
*/
// ================ END Controller ================ //



// ================ ElectricalLoad ================ //
/*
pybind11::class_<ElectricalLoad>(m, "ElectricalLoad")
    .def_readwrite("n_points", &ElectricalLoad::n_points)
    .def_readwrite("max_load_kW", &ElectricalLoad::max_load_kW)
    .def_readwrite("mean_load_kW", &ElectricalLoad::mean_load_kW)
    .def_readwrite("min_load_kW", &ElectricalLoad::min_load_kW)
    .def_readwrite("dt_vec_hrs", &ElectricalLoad::dt_vec_hrs)
    .def_readwrite("load_vec_kW", &ElectricalLoad::load_vec_kW)
    .def_readwrite("time_vec_hrs", &ElectricalLoad::time_vec_hrs)

    .def(pybind11::init<std::string>());
*/
// ================ END ElectricalLoad ================ //



// ================ Model ================ //
/*
pybind11::class_<Model>(m, "Model")
    .def(
        pybind11::init<
            ElectricalLoad*,
            RenewableResources*
        >()
    );
*/
// ================ END Model ================ //



// ================ RenewableResources ================ //
/*
pybind11::class_<RenewableResources>(m, "RenewableResources")
    .def(pybind11::init());
    /*
    .def(pybind11::init<>());
    */
*/
// ================ END RenewableResources ================ //

}   /* 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.

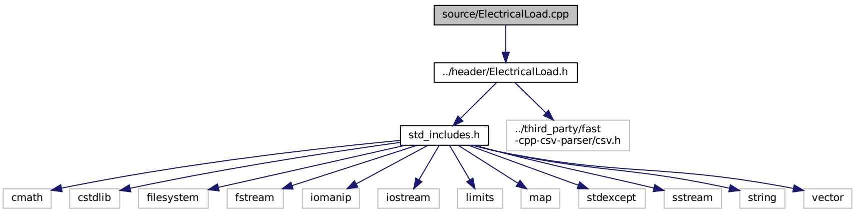
A class which contains a various dispatch control logic. Intended to serve as a component class of Controller.

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.

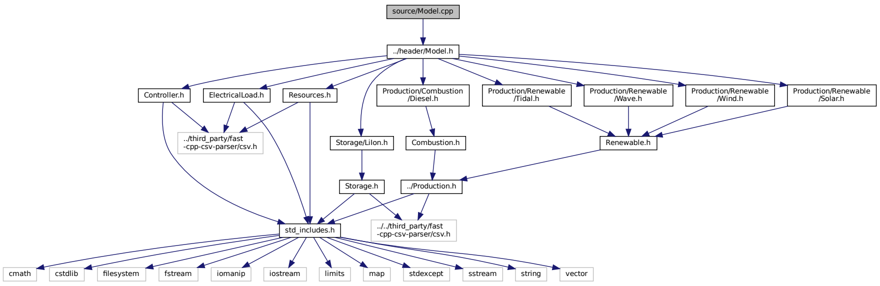
A class which contains time and electrical load data. Intended to serve as a component class of Model.

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.

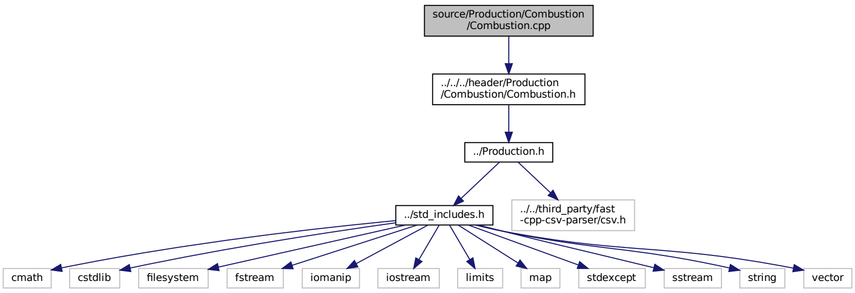
A container class which forms the centre of PGMcpp. The Model class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes.

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.

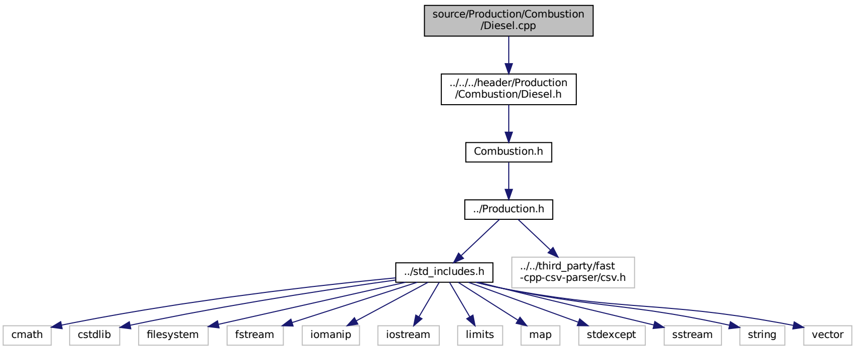
The root of the Combustion branch of the Production hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

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.

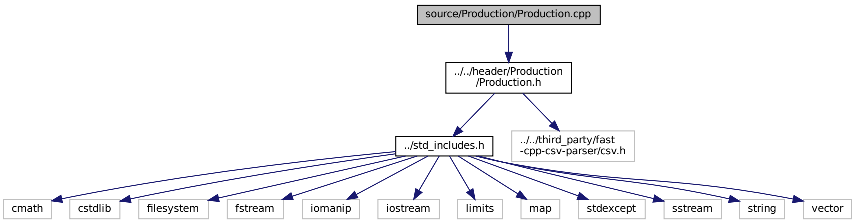
A derived class of the Combustion branch of Production which models production using a diesel generator.

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.

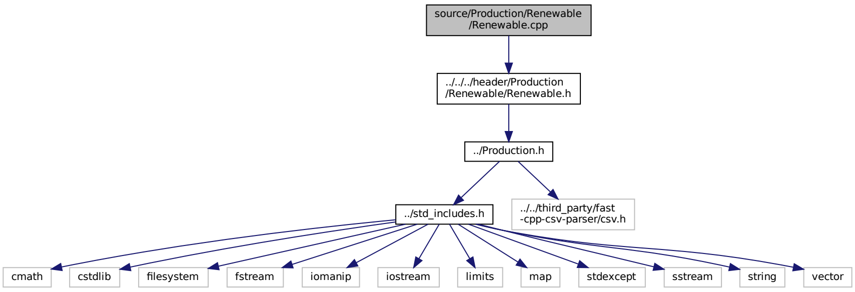
The base class of the Production hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise.

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.

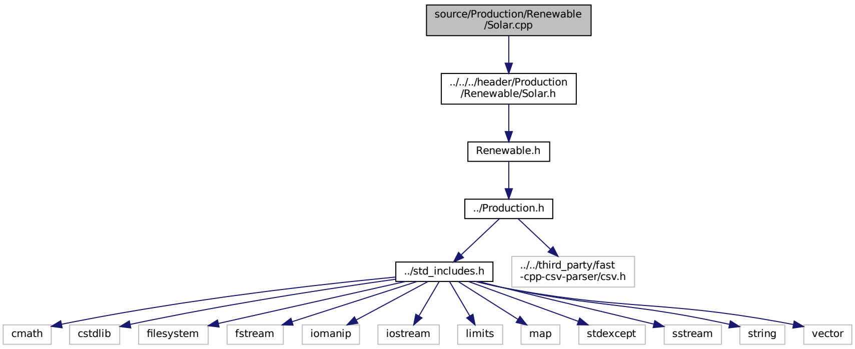
The root of the Renewable branch of the Production hierarchy. This branch contains derived classes which model the renewable production of energy.

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:


source/Production/Renewable/Solar.cpp
../../../header/Production/Renewable/Solar.h
Renewable.h
../Production.h
../std_includes.h
../../third_party/fast-cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

5.24.1 Detailed Description

Implementation file for the Solar class.

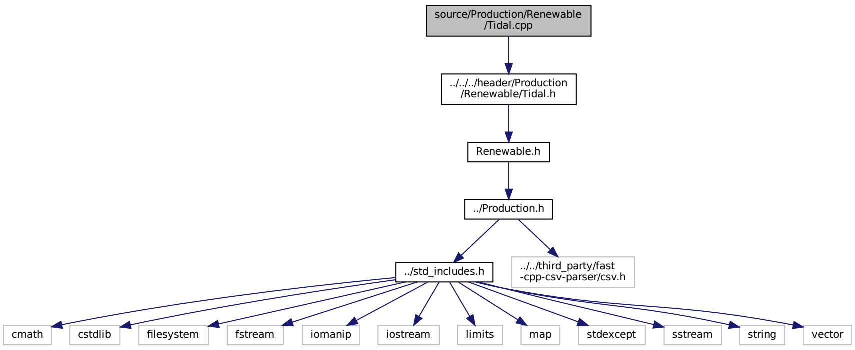
A derived class of the Renewable branch of Production which models solar production.

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:


source/Production/Renewable/Tidal.cpp
../../../header/Production/Renewable/Tidal.h
Renewable.h
../Production.h
../std_includes.h
../../third_party/fast-cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

5.25.1 Detailed Description

Implementation file for the Tidal class.

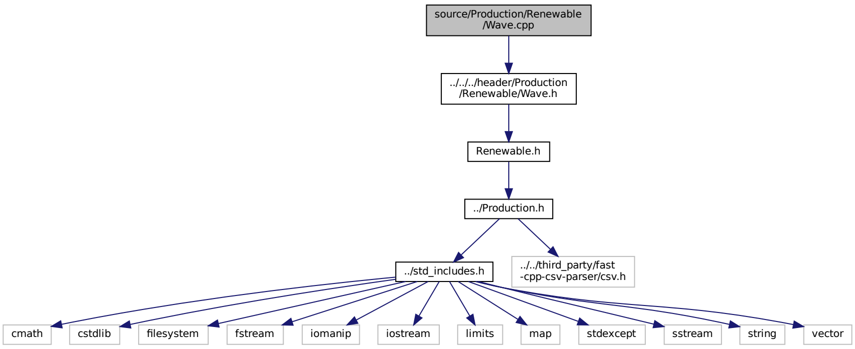
A derived class of the Renewable branch of Production which models tidal production.

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:


source/Production/Renewable /Wave.cpp
../../../header/Production /Renewable/Wave.h
Renewable.h
../Production.h
../std_includes.h
../../third_party/fast -cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

5.26.1 Detailed Description

Implementation file for the Wave class.

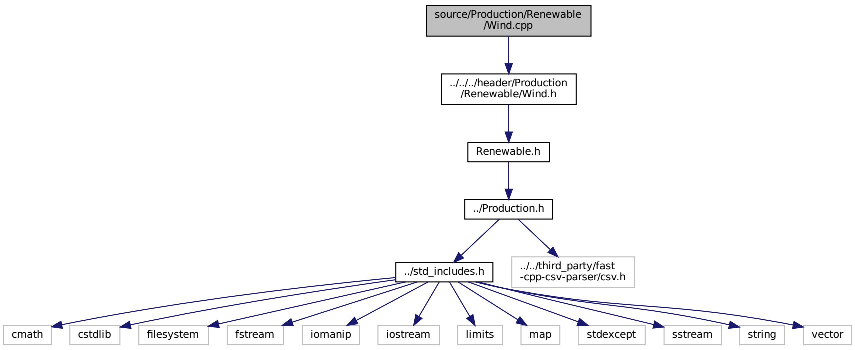
A derived class of the Renewable branch of Production which models wave production.

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:


source/Production/Renewable /Wind.cpp
../../../header/Production /Renewable/Wind.h
Renewable.h
../Production.h
../std_includes.h
../../third_party/fast -cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

5.27.1 Detailed Description

Implementation file for the Wind class.

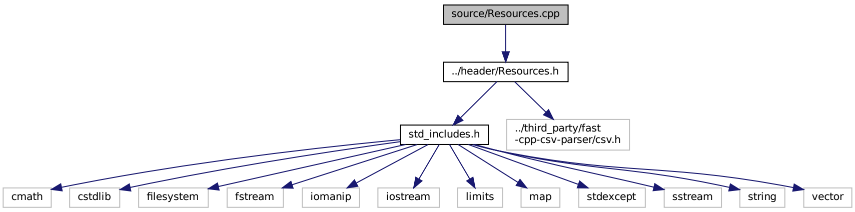
A derived class of the Renewable branch of Production which models wind production.

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.

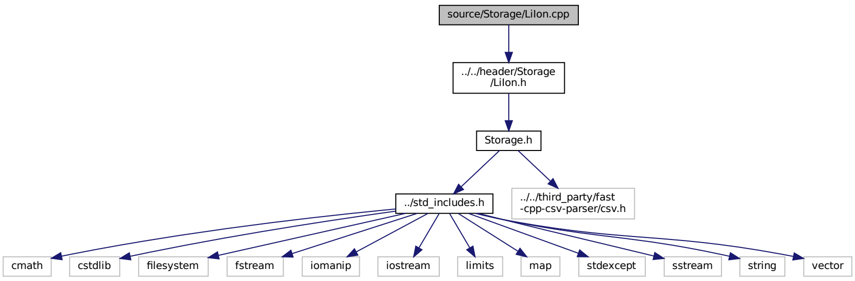
A class which contains renewable resource data. Intended to serve as a component class of Model.

5.29 source/Storage/LiIon.cpp File Reference

Implementation file for the LiIon class.

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

Include dependency graph for LiIon.cpp:



5.29.1 Detailed Description

Implementation file for the LiIon class.

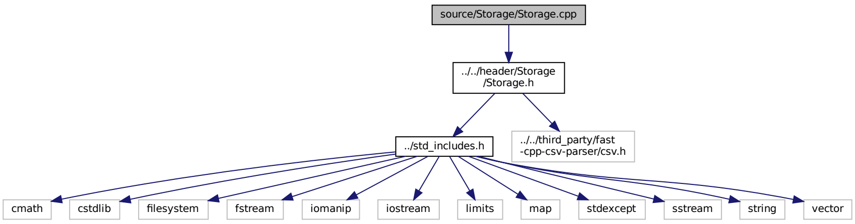
A derived class of Storage which models energy storage by way of lithium-ion batteries.

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.

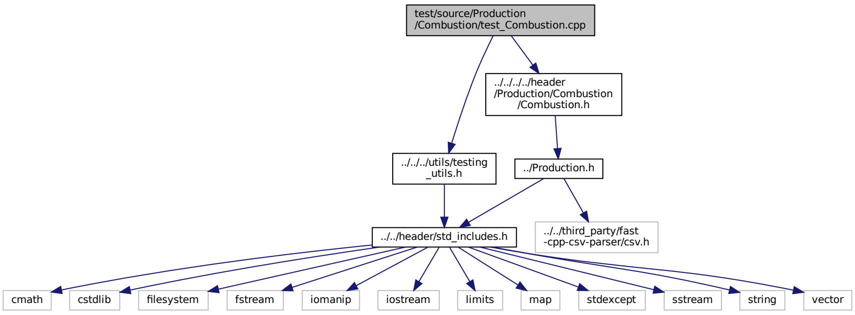
The base class of the Storage hierarchy. This hierarchy contains derived classes which model the storage of energy.

5.31 test/source/Production/Combustion/test_Combustion.cpp File Reference

Testing suite for Combustion class.

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

Include dependency graph for test_Combustion.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.31.1 Detailed Description

Testing suite for Combustion class.

A suite of tests for the Combustion class.

5.31.2 Function Documentation

5.31.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

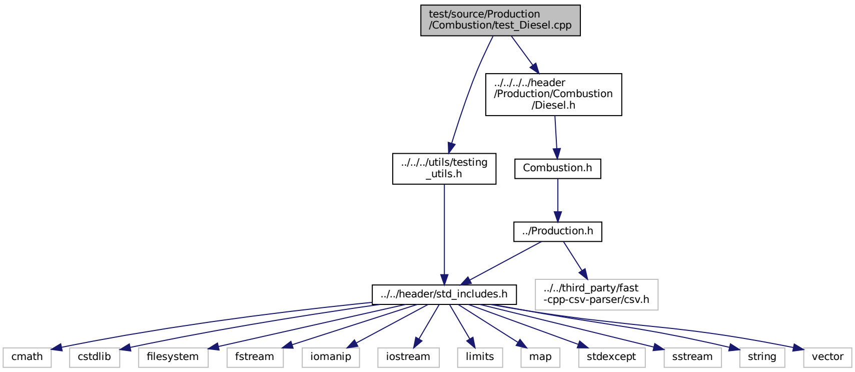
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Combustion");

    srand(time(NULL));


    try {
        //  1. construction
        CombustionInputs combustion_inputs;

        Combustion test_combustion(8760, combustion_inputs);


        //  2. test structure attributes
        testTruth(
            not combustion_inputs.production_inputs.print_flag,
            __FILE__,
            __LINE__
        );


        //  3. test post-construction attributes
        testFloatEquals(
            test_combustion.fuel_consumption_vec_L.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.fuel_cost_vec.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.CO2_emissions_vec_kg.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.CO_emissions_vec_kg.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.NOx_emissions_vec_kg.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.SOx_emissions_vec_kg.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.CH4_emissions_vec_kg.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_combustion.PM_emissions_vec_kg.size(),
            8760,
            __FILE__,
            __LINE__
        );
    }

    catch (...) {
        //...

        printGold(" .............. ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" .............. ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.32 test/source/Production/Combustion/test_Diesel.cpp File Reference

Testing suite for Diesel class.

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

Include dependency graph for test_Diesel.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.32.1 Detailed Description

Testing suite for Diesel class.

A suite of tests for the Diesel class.

5.32.2 Function Documentation

5.32.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

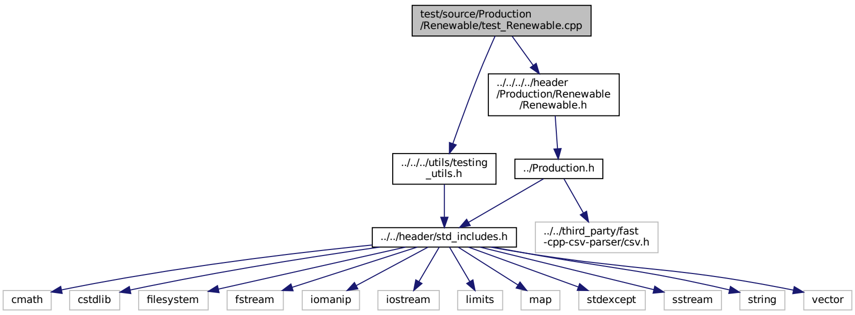
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Combustion <-- Diesel");

    srand(time(NULL));


    try {
        //  1. construction
        bool error_flag = true;
        try {
            DieselInputs bad_diesel_inputs;
            bad_diesel_inputs.fuel_cost_L = -1;

            Diesel bad_diesel(8760, bad_diesel_inputs);

            error_flag = false;
        } catch (...) {
            // Task failed successfully! =P
        }
        if (not error_flag) {
            expectedErrorNotDetected(__FILE__, __LINE__);
        }


        DieselInputs diesel_inputs;

        Diesel test_diesel(8760, diesel_inputs);

        //  2. test structure attributes
        testTruth(
            not diesel_inputs.combustion_inputs.production_inputs.print_flag,
            __FILE__,
            __LINE__
        );


        //  3. test post-construction attributes
        testFloatEquals(
            test_diesel.linear_fuel_slope_LkWh,
            0.265675,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_diesel.linear_fuel_intercept_LkWh,
            0.026676,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_diesel.capital_cost,
            67846.467018,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_diesel.operation_maintenance_cost_kWh,
            0.038027,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_diesel.minimum_load_ratio,
            0.2,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_diesel.minimum_runtime_hrs,
            1,
            __FILE__,
            __LINE__
        );


        //  4. test methods

        testFloatEquals(
            test_diesel.requestProductionkW(0, 1, 2 * test_diesel.capacity_kW),
            test_diesel.capacity_kW,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_diesel.requestProductionkW(
                0,
                1,
                0.5 * test_diesel.minimum_load_ratio * test_diesel.capacity_kW
            ),
            test_diesel.minimum_load_ratio * test_diesel.capacity_kW,
            __FILE__,
            __LINE__
        );
    }

    catch (...) {
        //...

        printGold(" ... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}    /* main() */
```

5.33 test/source/Production/Renewable/test_Renewable.cpp File Reference

Testing suite for Renewable class.

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

Include dependency graph for test_Renewable.cpp:


test/source/Production/Renewable/test_Renewable.cpp
../../../../header/Production/Renewable/Renewable.h
../../../utils/testing_utils.h
../Production.h
../../header/std_includes.h
../../third_party/fast-cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

Functions

- int main (int argc, char ∗∗argv)

5.33.1 Detailed Description

Testing suite for Renewable class.

A suite of tests for the Renewable class.

5.33.2 Function Documentation

5.33.2.1 main()

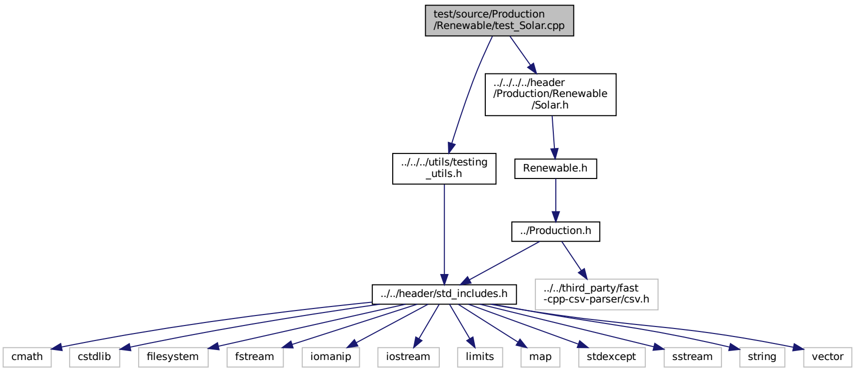
```
int main (
                int argc,
                char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Renewable");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ............... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ............... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.34 test/source/Production/Renewable/test_Solar.cpp File Reference

Testing suite for Solar class.

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

Include dependency graph for test_Solar.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.34.1 Detailed Description

Testing suite for Solar class.

A suite of tests for the Solar class.

5.34.2 Function Documentation

5.34.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

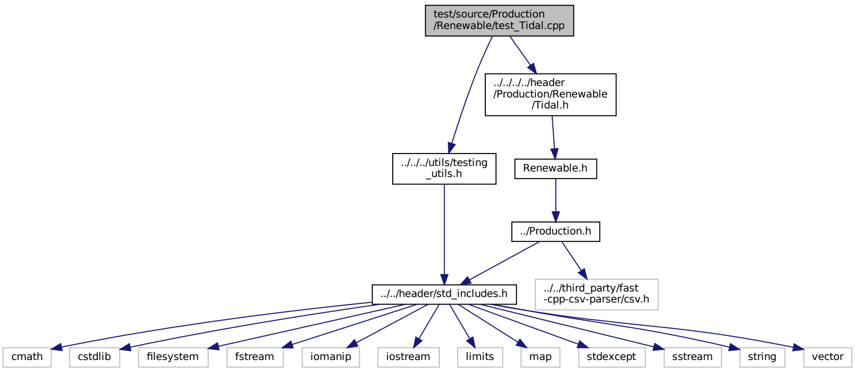
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Renewable <-- Solar");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ..... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ..... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.35 test/source/Production/Renewable/test_Tidal.cpp File Reference

Testing suite for Tidal class.

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

Include dependency graph for test_Tidal.cpp:


test/source/Production/Renewable/test_Tidal.cpp
../../../../header/Production/Renewable/Tidal.h
../../../utils/testing_utils.h
Renewable.h
../Production.h
../../header/std_includes.h
../../third_party/fast-cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

Functions

- int main (int argc, char **argv)

5.35.1 Detailed Description

Testing suite for Tidal class.

A suite of tests for the Tidal class.

5.35.2 Function Documentation

5.35.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

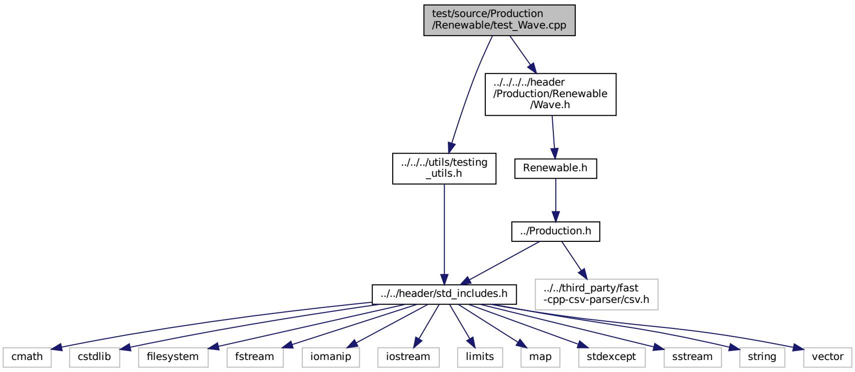
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Renewable <-- Tidal");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ..... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ..... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.36 test/source/Production/Renewable/test_Wave.cpp File Reference

Testing suite for Wave class.

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

Include dependency graph for test_Wave.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.36.1 Detailed Description

Testing suite for Wave class.

A suite of tests for the Wave class.

5.36.2 Function Documentation

5.36.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

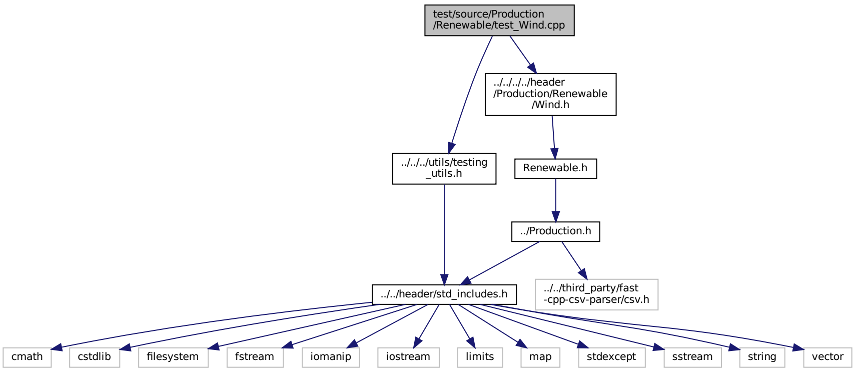
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Renewable <-- Wave");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ...... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ...... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.37 test/source/Production/Renewable/test_Wind.cpp File Reference

Testing suite for Wind class.

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

Include dependency graph for test_Wind.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.37.1 Detailed Description

Testing suite for Wind class.

A suite of tests for the Wind class.

5.37.2 Function Documentation

5.37.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

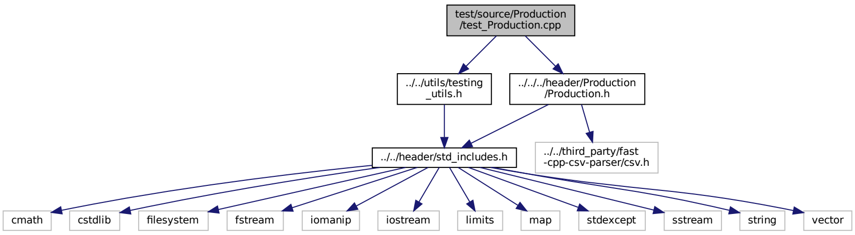
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Production <-- Renewable <-- Wind");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ...... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ...... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.38 test/source/Production/test_Production.cpp File Reference

Testing suite for Production class.

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

Include dependency graph for test_Production.cpp:



Functions

- int main (int argc, char **argv)

5.38.1 Detailed Description

Testing suite for Production class.

A suite of tests for the Production class.

5.38.2 Function Documentation

5.38.2.1 main()

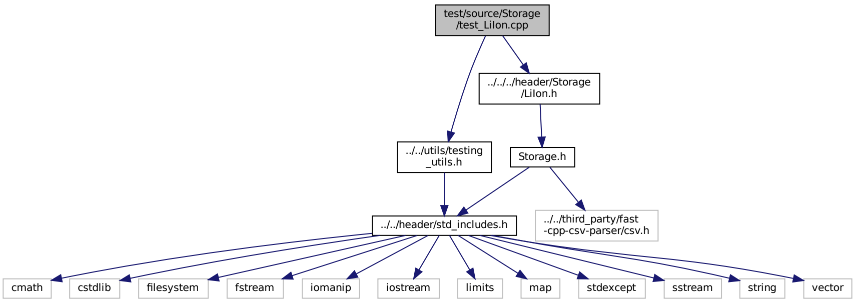
```
int main (
                int argc,
                char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\n\tTesting Production");

    srand(time(NULL));


    try {
        //  1. construction
        bool error_flag = true;
        try {
            ProductionInputs production_inputs;

            Production bad_production(0, production_inputs);

            error_flag = false;
        } catch (...) {
            // Task failed successfully! =P
        }
        if (not error_flag) {
            expectedErrorNotDetected(__FILE__, __LINE__);
        }


        ProductionInputs production_inputs;

        Production test_production(8760, production_inputs);


        //  2. test structure attributes
        testTruth(
            not production_inputs.print_flag,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            production_inputs.nominal_inflation_annual,
            0.02,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            production_inputs.nominal_discount_annual,
            0.04,
            __FILE__,
            __LINE__
        );


        //  3. test post-construction attributes
        testFloatEquals(
            test_production.n_points,
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.capacity_kW,
            100,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.real_discount_annual,
            0.0196078431372549,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.production_vec_kW.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.dispatch_vec_kW.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.storage_vec_kW.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.curtailment_vec_kW.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.capital_cost_vec.size(),
            8760,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            test_production.operation_maintenance_cost_vec.size(),
            8760,
            __FILE__,
            __LINE__
        );
    }

    catch (...) {
        //...

        printGold(" ............................ ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ............................ ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.39 test/source/Storage/test_LiIon.cpp File Reference

Testing suite for LiIon class.

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

Include dependency graph for test_LiIon.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.39.1 Detailed Description

Testing suite for LiIon class.

A suite of tests for the LiIon class.

5.39.2 Function Documentation

5.39.2.1 main()

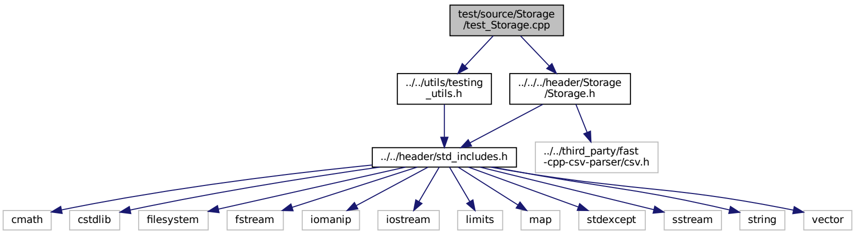
```
int main (
              int argc,
              char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Storage <-- LiIon");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ...................... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ...................... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.40 test/source/Storage/test_Storage.cpp File Reference

Testing suite for Storage class.

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

Include dependency graph for test_Storage.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.40.1 Detailed Description

Testing suite for Storage class.

A suite of tests for the Storage class.

5.40.2 Function Documentation

5.40.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

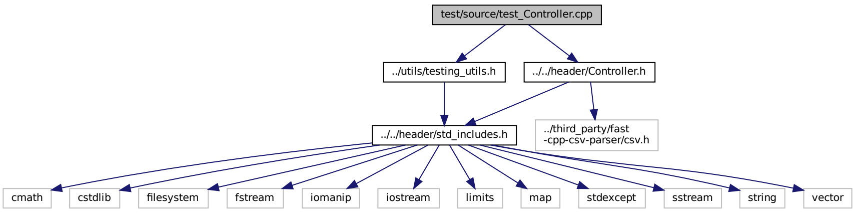
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Storage");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ...................... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ................................ ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.41 test/source/test_Controller.cpp File Reference

Testing suite for Controller class.

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

Include dependency graph for test_Controller.cpp:



Functions

- int main (int argc, char **argv)

5.41.1 Detailed Description

Testing suite for Controller class.

A suite of tests for the Controller class.

5.41.2 Function Documentation

5.41.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

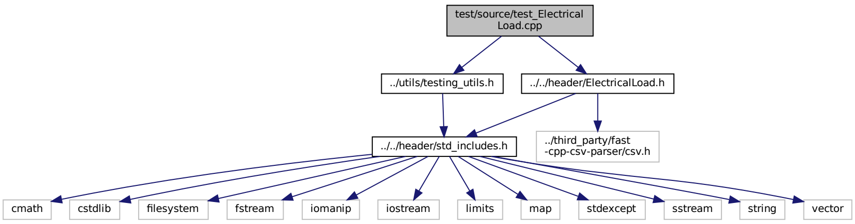
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Controller");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ............................. ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ............................. ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.42 test/source/test_ElectricalLoad.cpp File Reference

Testing suite for ElectricalLoad class.

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

Include dependency graph for test_ElectricalLoad.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.42.1 Detailed Description

Testing suite for ElectricalLoad class.

A suite of tests for the ElectricalLoad class.

5.42.2 Function Documentation

5.42.2.1 main()

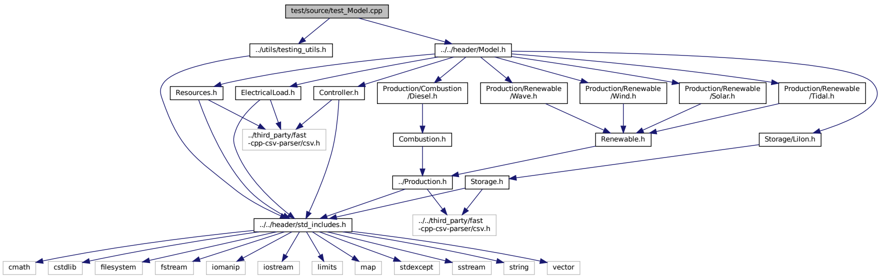
```
int main (
            int argc,
            char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting ElectricalLoad");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ......................... ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ......................... ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.43 test/source/test_Model.cpp File Reference

Testing suite for Model class.

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

Include dependency graph for test_Model.cpp:


test/source/test_Model.cpp
../utils/testing_utils.h
../../header/Model.h
Resources.h
ElectricalLoad.h
Controller.h
Production/Combustion /Diesel.h
Production/Renewable /Wave.h
Production/Renewable /Wind.h
Production/Renewable /Solar.h
Production/Renewable /Tidal.h
../third_party/fast -cpp-csv-parser/csv.h
Combustion.h
Renewable.h
Storage/LiIon.h
../Production.h
Storage.h
../../header/std_includes.h
../../third_party/fast -cpp-csv-parser/csv.h
cmath
cstdlib
filesystem
fstream
iomanip
iostream
limits
map
stdexcept
sstream
string
vector

Functions

- int main (int argc, char ∗∗argv)

5.43.1 Detailed Description

Testing suite for Model class.

A suite of tests for the Model class.

5.43.2 Function Documentation

5.43.2.1 main()

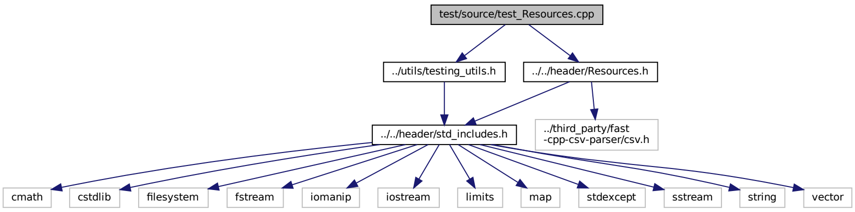
```
int main (
            int argc,
            char ** argv )
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Model");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ................................. ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ................................. ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.44 test/source/test_Resources.cpp File Reference

Testing suite for Resources class.

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

Include dependency graph for test_Resources.cpp:



Functions

- int main (int argc, char ∗∗argv)

5.44.1 Detailed Description

Testing suite for Resources class.

A suite of tests for the Resources class.

5.44.2 Function Documentation

5.44.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

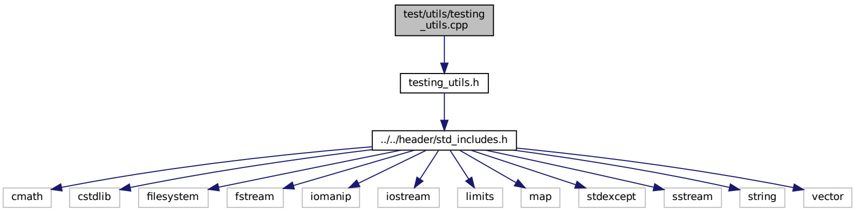
```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting Resources");

    srand(time(NULL));


    try {
        //...
    }

    catch (...) {
        //...

        printGold(" ............................. ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    printGold(" ............................. ");
    printGreen("PASS");
    std::cout « std::endl;
    return 0;
}   /* main() */
```

5.45 test/utils/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 $x >= y$.
- 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 $x <= y$.
- 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.

5.45.1 Detailed Description

Header file for various PGMcpp testing utilities.

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

5.45.2 Function Documentation

5.45.2.1 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

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

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

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* expectedErrorNotDetected() */
```

5.45.2.2 printGold()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

5.45.2.3 printGreen()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

5.45.2.4 printRed()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

5.45.2.5 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

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

```
{
    if (fabs(x - y) <= FLOAT_TOLERANCE) {
        return;
    }

    std::string error_str = "ERROR: testFloatEquals():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " and ";
    error_str += std::to_string(y);
    error_str += " are not equal to within +/- ";
    error_str += std::to_string(FLOAT_TOLERANCE);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testFloatEquals() */
```

5.45.2.6 testGreaterThan()

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

Tests if x > y.

Parameters

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

```
{
    if (x > y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThan() */
```

5.45.2.7 testGreaterThanOrEqualTo()

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

Tests if x >= y.

Parameters

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

```
{
    if (x >= y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThanOrEqualTo() */
```

5.45.2.8 testLessThan()

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

Tests if x < y.

Parameters

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

```
{
    if (x < y) {
        return;
    }

    std::string error_str = "ERROR: testLessThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThan() */
```

5.45.2.9 testLessThanOrEqualTo()

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

Tests if x <= y.

Parameters

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

```
{
    if (x <= y) {
        return;
    }

    std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThanOrEqualTo() */
```

5.45.2.10 testTruth()

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

Tests if the given statement is true.

Parameters

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

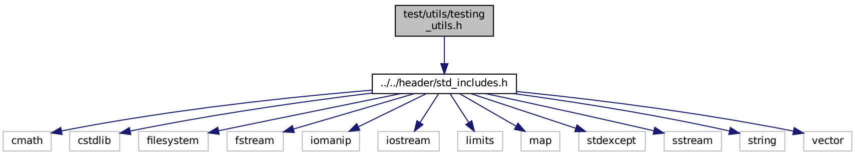
```
{
    if (statement) {
        return;
    }

    std::string error_str = "ERROR: testTruth():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += "Given statement is not true";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testTruth() */
```

5.46 test/utils/testing_utils.h File Reference

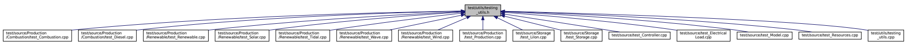
Header file for various PGMcpp testing utilities.

`#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 >= y.
- void testLessThan (double, double, std::string, int)

  Tests if x < y.
- void testLessThanOrEqualTo (double, double, std::string, int)

  Tests if x <= 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.

5.46.1 Detailed Description

Header file for various PGMcpp testing utilities.

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

5.46.2 Macro Definition Documentation

5.46.2.1 FLOAT_TOLERANCE

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

A tolerance for application to floating point equality tests.

5.46.3 Function Documentation

5.46.3.1 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

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

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

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* expectedErrorNotDetected() */
```

5.46.3.2 printGold()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

5.46.3.3 printGreen()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

5.46.3.4 printRed()

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

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

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

5.46.3.5 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

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

```
{
    if (fabs(x - y) <= FLOAT_TOLERANCE) {
        return;
    }

    std::string error_str = "ERROR: testFloatEquals():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " and ";
    error_str += std::to_string(y);
    error_str += " are not equal to within +/- ";
    error_str += std::to_string(FLOAT_TOLERANCE);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testFloatEquals() */
```

5.46.3.6 testGreaterThan()

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

Tests if x > y.

Parameters

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

```
{
    if (x > y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThan() */
```

5.46.3.7 testGreaterThanOrEqualTo()

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

Tests if x >= y.

Parameters

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

```
{
    if (x >= y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThanOrEqualTo() */
```

5.46.3.8 testLessThan()

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

Tests if x $<$ y.

Parameters

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

```
{
    if (x < y) {
        return;
    }

    std::string error_str = "ERROR: testLessThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThan() */
```

5.46.3.9 testLessThanOrEqualTo()

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

Tests if x <= y.

Parameters

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

```
{
    if (x <= y) {
        return;
    }

    std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThanOrEqualTo() */
```

5.46.3.10 testTruth()

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

Tests if the given statement is true.

Parameters

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

```
{
    if (statement) {
        return;
    }

    std::string error_str = "ERROR: testTruth():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += "Given statement is not true";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testTruth() */
```

Index