

PGMcpp: PRIMED Grid Modelling (in C++)

Generated by Doxygen 1.9.1

| | |
|--|----------|
| 1 Hierarchical Index | 1 |
| 1.1 Class Hierarchy | 1 |
| 2 Class Index | 3 |
| 2.1 Class List | 3 |
| 3 File Index | 5 |
| 3.1 File List | 5 |
| 4 Class Documentation | 7 |
| 4.1 Combustion Class Reference | 7 |
| 4.1.1 Detailed Description | 9 |
| 4.1.2 Constructor & Destructor Documentation | 9 |
| 4.1.2.1 Combustion() [1/2] | 9 |
| 4.1.2.2 Combustion() [2/2] | 9 |
| 4.1.2.3 ~Combustion() | 10 |
| 4.1.3 Member Function Documentation | 10 |
| 4.1.3.1 commit() | 10 |
| 4.1.3.2 getEmissionskg() | 11 |
| 4.1.3.3 getFuelConsumptionL() | 11 |
| 4.1.3.4 requestProductionkW() | 12 |
| 4.1.4 Member Data Documentation | 12 |
| 4.1.4.1 CH4_emissions_vec_kg | 12 |
| 4.1.4.2 CO2_emissions_vec_kg | 12 |
| 4.1.4.3 CO_emissions_vec_kg | 12 |
| 4.1.4.4 fuel_consumption_vec_L | 13 |
| 4.1.4.5 fuel_cost_vec | 13 |
| 4.1.4.6 linear_fuel_intercept_LkWh | 13 |
| 4.1.4.7 linear_fuel_slope_LkWh | 13 |
| 4.1.4.8 NOx_emissions_vec_kg | 13 |
| 4.1.4.9 PM_emissions_vec_kg | 13 |
| 4.1.4.10 SOx_emissions_vec_kg | 14 |
| 4.2 CombustionInputs Struct Reference | 14 |
| 4.2.1 Detailed Description | 14 |
| 4.2.2 Member Data Documentation | 14 |
| 4.2.2.1 production_inputs | 15 |
| 4.3 Controller Class Reference | 15 |
| 4.3.1 Detailed Description | 15 |
| 4.3.2 Constructor & Destructor Documentation | 15 |
| 4.3.2.1 Controller() | 15 |
| 4.3.2.2 ~Controller() | 16 |
| 4.4 Diesel Class Reference | 16 |
| 4.4.1 Detailed Description | 18 |

| | |
|--|----|
| 4.4.2 Constructor & Destructor Documentation | 18 |
| 4.4.2.1 Diesel() [1/2] | 18 |
| 4.4.2.2 Diesel() [2/2] | 18 |
| 4.4.2.3 ~Diesel() | 19 |
| 4.4.3 Member Function Documentation | 19 |
| 4.4.3.1 commit() | 19 |
| 4.4.3.2 requestProductionkW() | 20 |
| 4.4.4 Member Data Documentation | 21 |
| 4.4.4.1 CH4_emissions_intensity_kgL | 21 |
| 4.4.4.2 CO2_emissions_intensity_kgL | 21 |
| 4.4.4.3 CO_emissions_intensity_kgL | 21 |
| 4.4.4.4 fuel_cost_L | 21 |
| 4.4.4.5 minimum_load_ratio | 21 |
| 4.4.4.6 minimum_runtime_hrs | 22 |
| 4.4.4.7 NOx_emissions_intensity_kgL | 22 |
| 4.4.4.8 PM_emissions_intensity_kgL | 22 |
| 4.4.4.9 SOx_emissions_intensity_kgL | 22 |
| 4.4.4.10 time_since_last_start_hrs | 22 |
| 4.5 DieselInputs Struct Reference | 23 |
| 4.5.1 Detailed Description | 24 |
| 4.5.2 Member Data Documentation | 24 |
| 4.5.2.1 capital_cost | 24 |
| 4.5.2.2 CH4_emissions_intensity_kgL | 24 |
| 4.5.2.3 CO2_emissions_intensity_kgL | 24 |
| 4.5.2.4 CO_emissions_intensity_kgL | 25 |
| 4.5.2.5 combustion_inputs | 25 |
| 4.5.2.6 fuel_cost_L | 25 |
| 4.5.2.7 linear_fuel_intercept_LkWh | 25 |
| 4.5.2.8 linear_fuel_slope_LkWh | 25 |
| 4.5.2.9 minimum_load_ratio | 25 |
| 4.5.2.10 minimum_runtime_hrs | 26 |
| 4.5.2.11 NOx_emissions_intensity_kgL | 26 |
| 4.5.2.12 operation_maintenance_cost_kWh | 26 |
| 4.5.2.13 PM_emissions_intensity_kgL | 26 |
| 4.5.2.14 SOx_emissions_intensity_kgL | 26 |
| 4.6 ElectricalLoad Class Reference | 26 |
| 4.6.1 Detailed Description | 27 |
| 4.6.2 Constructor & Destructor Documentation | 27 |
| 4.6.2.1 ElectricalLoad() | 27 |
| 4.6.2.2 ~ElectricalLoad() | 27 |
| 4.7 Emissions Struct Reference | 27 |
| 4.7.1 Detailed Description | 28 |

| | |
|---|----|
| 4.7.2 Member Data Documentation | 28 |
| 4.7.2.1 CH4_kg | 28 |
| 4.7.2.2 CO2_kg | 28 |
| 4.7.2.3 CO_kg | 28 |
| 4.7.2.4 NOx_kg | 29 |
| 4.7.2.5 PM_kg | 29 |
| 4.7.2.6 SOx_kg | 29 |
| 4.8 Lilon Class Reference | 29 |
| 4.8.1 Detailed Description | 30 |
| 4.8.2 Constructor & Destructor Documentation | 30 |
| 4.8.2.1 Lilon() | 30 |
| 4.8.2.2 ~Lilon() | 31 |
| 4.9 Model Class Reference | 31 |
| 4.9.1 Detailed Description | 32 |
| 4.9.2 Constructor & Destructor Documentation | 32 |
| 4.9.2.1 Model() | 32 |
| 4.9.2.2 ~Model() | 32 |
| 4.9.3 Member Data Documentation | 33 |
| 4.9.3.1 combustion_ptr_vec | 33 |
| 4.9.3.2 controller | 33 |
| 4.9.3.3 electrical_load | 33 |
| 4.9.3.4 renewable_ptr_vec | 33 |
| 4.9.3.5 resources | 33 |
| 4.9.3.6 storage_ptr_vec | 34 |
| 4.10 Production Class Reference | 34 |
| 4.10.1 Detailed Description | 35 |
| 4.10.2 Constructor & Destructor Documentation | 36 |
| 4.10.2.1 Production() [1/2] | 36 |
| 4.10.2.2 Production() [2/2] | 36 |
| 4.10.2.3 ~Production() | 37 |
| 4.10.3 Member Function Documentation | 37 |
| 4.10.3.1 commit() | 37 |
| 4.10.4 Member Data Documentation | 38 |
| 4.10.4.1 capacity_kW | 38 |
| 4.10.4.2 capital_cost | 38 |
| 4.10.4.3 capital_cost_vec | 38 |
| 4.10.4.4 curtailment_vec_kW | 39 |
| 4.10.4.5 dispatch_vec_kW | 39 |
| 4.10.4.6 is_running | 39 |
| 4.10.4.7 is_running_vec | 39 |
| 4.10.4.8 is_sunk | 39 |
| 4.10.4.9 levlized_cost_of_energy_kWh | 39 |

| | |
|---|----|
| 4.10.4.10 n_points | 40 |
| 4.10.4.11 n_starts | 40 |
| 4.10.4.12 net_present_cost | 40 |
| 4.10.4.13 operation_maintenance_cost_kWh | 40 |
| 4.10.4.14 operation_maintenance_cost_vec | 40 |
| 4.10.4.15 print_flag | 40 |
| 4.10.4.16 production_vec_kW | 41 |
| 4.10.4.17 real_discount_annual | 41 |
| 4.10.4.18 running_hours | 41 |
| 4.10.4.19 storage_vec_kW | 41 |
| 4.11 ProductionInputs Struct Reference | 41 |
| 4.11.1 Detailed Description | 42 |
| 4.11.2 Member Data Documentation | 42 |
| 4.11.2.1 capacity_kW | 42 |
| 4.11.2.2 is_sunk | 42 |
| 4.11.2.3 nominal_discount_annual | 42 |
| 4.11.2.4 nominal_inflation_annual | 43 |
| 4.11.2.5 print_flag | 43 |
| 4.12 Renewable Class Reference | 43 |
| 4.12.1 Detailed Description | 44 |
| 4.12.2 Constructor & Destructor Documentation | 44 |
| 4.12.2.1 Renewable() | 44 |
| 4.12.2.2 ~Renewable() | 45 |
| 4.13 Resources Class Reference | 45 |
| 4.13.1 Detailed Description | 45 |
| 4.13.2 Constructor & Destructor Documentation | 45 |
| 4.13.2.1 Resources() | 45 |
| 4.13.2.2 ~Resources() | 46 |
| 4.14 Solar Class Reference | 46 |
| 4.14.1 Detailed Description | 47 |
| 4.14.2 Constructor & Destructor Documentation | 47 |
| 4.14.2.1 Solar() | 47 |
| 4.14.2.2 ~Solar() | 48 |
| 4.15 Storage Class Reference | 48 |
| 4.15.1 Detailed Description | 48 |
| 4.15.2 Constructor & Destructor Documentation | 49 |
| 4.15.2.1 Storage() | 49 |
| 4.15.2.2 ~Storage() | 49 |
| 4.16 Tidal Class Reference | 49 |
| 4.16.1 Detailed Description | 50 |
| 4.16.2 Constructor & Destructor Documentation | 50 |
| 4.16.2.1 Tidal() | 50 |

| | |
|--|-----------|
| 4.16.2.2 ~Tidal() | 51 |
| 4.17 Wave Class Reference | 51 |
| 4.17.1 Detailed Description | 52 |
| 4.17.2 Constructor & Destructor Documentation | 52 |
| 4.17.2.1 Wave() | 52 |
| 4.17.2.2 ~Wave() | 53 |
| 4.18 Wind Class Reference | 53 |
| 4.18.1 Detailed Description | 54 |
| 4.18.2 Constructor & Destructor Documentation | 54 |
| 4.18.2.1 Wind() | 54 |
| 4.18.2.2 ~Wind() | 55 |
| 5 File Documentation | 57 |
| 5.1 header/Controller.h File Reference | 57 |
| 5.1.1 Detailed Description | 58 |
| 5.2 header/ElectricalLoad.h File Reference | 58 |
| 5.2.1 Detailed Description | 58 |
| 5.3 header/Model.h File Reference | 59 |
| 5.3.1 Detailed Description | 59 |
| 5.4 header/Production/Combustion/Combustion.h File Reference | 60 |
| 5.4.1 Detailed Description | 60 |
| 5.4.2 Enumeration Type Documentation | 61 |
| 5.4.2.1 CombustionType | 61 |
| 5.5 header/Production/Combustion/Diesel.h File Reference | 61 |
| 5.5.1 Detailed Description | 62 |
| 5.6 header/Production/Production.h File Reference | 62 |
| 5.6.1 Detailed Description | 62 |
| 5.7 header/Production/Renewable/Renewable.h File Reference | 63 |
| 5.7.1 Detailed Description | 63 |
| 5.8 header/Production/Renewable/Solar.h File Reference | 63 |
| 5.8.1 Detailed Description | 64 |
| 5.9 header/Production/Renewable/Tidal.h File Reference | 64 |
| 5.9.1 Detailed Description | 65 |
| 5.10 header/Production/Renewable/Wave.h File Reference | 65 |
| 5.10.1 Detailed Description | 66 |
| 5.11 header/Production/Renewable/Wind.h File Reference | 66 |
| 5.11.1 Detailed Description | 67 |
| 5.12 header/Resources.h File Reference | 67 |
| 5.12.1 Detailed Description | 68 |
| 5.13 header/std_includes.h File Reference | 68 |
| 5.13.1 Detailed Description | 68 |
| 5.14 header/Storage/Lilon.h File Reference | 69 |

| | |
|---|----|
| 5.14.1 Detailed Description | 69 |
| 5.15 header/Storage/Storage.h File Reference | 69 |
| 5.15.1 Detailed Description | 70 |
| 5.16 pybindings/PYBIND11_PGM.cpp File Reference | 70 |
| 5.16.1 Detailed Description | 71 |
| 5.16.2 Function Documentation | 71 |
| 5.16.2.1 PYBIND11_MODULE() | 71 |
| 5.17 source/Controller.cpp File Reference | 72 |
| 5.17.1 Detailed Description | 72 |
| 5.18 source/ElectricalLoad.cpp File Reference | 73 |
| 5.18.1 Detailed Description | 73 |
| 5.19 source/Model.cpp File Reference | 73 |
| 5.19.1 Detailed Description | 73 |
| 5.20 source/Production/Combustion/Combustion.cpp File Reference | 74 |
| 5.20.1 Detailed Description | 74 |
| 5.21 source/Production/Combustion/Diesel.cpp File Reference | 74 |
| 5.21.1 Detailed Description | 75 |
| 5.22 source/Production/Production.cpp File Reference | 75 |
| 5.22.1 Detailed Description | 75 |
| 5.23 source/Production/Renewable/Renewable.cpp File Reference | 75 |
| 5.23.1 Detailed Description | 76 |
| 5.24 source/Production/Renewable/Solar.cpp File Reference | 76 |
| 5.24.1 Detailed Description | 76 |
| 5.25 source/Production/Renewable/Tidal.cpp File Reference | 76 |
| 5.25.1 Detailed Description | 77 |
| 5.26 source/Production/Renewable/Wave.cpp File Reference | 77 |
| 5.26.1 Detailed Description | 77 |
| 5.27 source/Production/Renewable/Wind.cpp File Reference | 77 |
| 5.27.1 Detailed Description | 78 |
| 5.28 source/Resources.cpp File Reference | 78 |
| 5.28.1 Detailed Description | 78 |
| 5.29 source/Storage/Lilon.cpp File Reference | 78 |
| 5.29.1 Detailed Description | 79 |
| 5.30 source/Storage/Storage.cpp File Reference | 79 |
| 5.30.1 Detailed Description | 79 |
| 5.31 test/source/Production/Combustion/test_Combustion.cpp File Reference | 79 |
| 5.31.1 Detailed Description | 80 |
| 5.31.2 Function Documentation | 80 |
| 5.31.2.1 main() | 80 |
| 5.32 test/source/Production/Combustion/test_Diesel.cpp File Reference | 81 |
| 5.32.1 Detailed Description | 82 |
| 5.32.2 Function Documentation | 82 |

| | |
|---|----|
| 5.32.2.1 main() | 82 |
| 5.33 test/source/Production/Renewable/test_Renewable.cpp File Reference | 83 |
| 5.33.1 Detailed Description | 84 |
| 5.33.2 Function Documentation | 84 |
| 5.33.2.1 main() | 84 |
| 5.34 test/source/Production/Renewable/test_Solar.cpp File Reference | 85 |
| 5.34.1 Detailed Description | 85 |
| 5.34.2 Function Documentation | 85 |
| 5.34.2.1 main() | 85 |
| 5.35 test/source/Production/Renewable/test_Tidal.cpp File Reference | 86 |
| 5.35.1 Detailed Description | 86 |
| 5.35.2 Function Documentation | 86 |
| 5.35.2.1 main() | 87 |
| 5.36 test/source/Production/Renewable/test_Wave.cpp File Reference | 87 |
| 5.36.1 Detailed Description | 88 |
| 5.36.2 Function Documentation | 88 |
| 5.36.2.1 main() | 88 |
| 5.37 test/source/Production/Renewable/test_Wind.cpp File Reference | 88 |
| 5.37.1 Detailed Description | 89 |
| 5.37.2 Function Documentation | 89 |
| 5.37.2.1 main() | 89 |
| 5.38 test/source/Production/test_Production.cpp File Reference | 90 |
| 5.38.1 Detailed Description | 90 |
| 5.38.2 Function Documentation | 90 |
| 5.38.2.1 main() | 90 |
| 5.39 test/source/Storage/test_Lilon.cpp File Reference | 92 |
| 5.39.1 Detailed Description | 92 |
| 5.39.2 Function Documentation | 93 |
| 5.39.2.1 main() | 93 |
| 5.40 test/source/Storage/test_Storage.cpp File Reference | 93 |
| 5.40.1 Detailed Description | 94 |
| 5.40.2 Function Documentation | 94 |
| 5.40.2.1 main() | 94 |
| 5.41 test/source/test_Controller.cpp File Reference | 94 |
| 5.41.1 Detailed Description | 95 |
| 5.41.2 Function Documentation | 95 |
| 5.41.2.1 main() | 95 |
| 5.42 test/source/test_ElectricalLoad.cpp File Reference | 95 |
| 5.42.1 Detailed Description | 96 |
| 5.42.2 Function Documentation | 96 |
| 5.42.2.1 main() | 96 |
| 5.43 test/source/test_Model.cpp File Reference | 97 |

| | |
|--|-----|
| 5.43.1 Detailed Description | 97 |
| 5.43.2 Function Documentation | 97 |
| 5.43.2.1 main() | 97 |
| 5.44 test/source/test_Resources.cpp File Reference | 98 |
| 5.44.1 Detailed Description | 98 |
| 5.44.2 Function Documentation | 98 |
| 5.44.2.1 main() | 99 |
| 5.45 test/utls/testing_utls.cpp File Reference | 99 |
| 5.45.1 Detailed Description | 100 |
| 5.45.2 Function Documentation | 100 |
| 5.45.2.1 expectedErrorNotDetected() | 100 |
| 5.45.2.2 printGold() | 101 |
| 5.45.2.3 printGreen() | 101 |
| 5.45.2.4 printRed() | 101 |
| 5.45.2.5 testFloatEquals() | 101 |
| 5.45.2.6 testGreaterThan() | 102 |
| 5.45.2.7 testGreaterThanOrEqualTo() | 103 |
| 5.45.2.8 testLessThan() | 103 |
| 5.45.2.9 testLessThanOrEqualTo() | 104 |
| 5.45.2.10 testTruth() | 105 |
| 5.46 test/utls/testing_utls.h File Reference | 105 |
| 5.46.1 Detailed Description | 106 |
| 5.46.2 Macro Definition Documentation | 106 |
| 5.46.2.1 FLOAT_TOLERANCE | 106 |
| 5.46.3 Function Documentation | 107 |
| 5.46.3.1 expectedErrorNotDetected() | 107 |
| 5.46.3.2 printGold() | 107 |
| 5.46.3.3 printGreen() | 107 |
| 5.46.3.4 printRed() | 108 |
| 5.46.3.5 testFloatEquals() | 108 |
| 5.46.3.6 testGreaterThan() | 109 |
| 5.46.3.7 testGreaterThanOrEqualTo() | 109 |
| 5.46.3.8 testLessThan() | 110 |
| 5.46.3.9 testLessThanOrEqualTo() | 111 |
| 5.46.3.10 testTruth() | 111 |

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

| | |
|----------------------------|----|
| CombustionInputs | 14 |
| Controller | 15 |
| DieselInputs | 23 |
| ElectricalLoad | 26 |
| Emissions | 27 |
| Model | 31 |
| Production | 34 |
| Combustion | 7 |
| Diesel | 16 |
| Renewable | 43 |
| Solar | 46 |
| Tidal | 49 |
| Wave | 51 |
| Wind | 53 |
| ProductionInputs | 41 |
| Resources | 45 |
| Storage | 48 |
| Lilon | 29 |

Chapter 2

Class Index

2.1 Class List

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

| | | |
|----------------------------------|--|----|
| 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 | 7 |
| 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 . . . | 14 |
| Controller | A class which contains a various dispatch control logic. Intended to serve as a component class of Model | 15 |
| Diesel | A derived class of the Combustion branch of Production which models production using a diesel generator | 16 |
| 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 . . . | 23 |
| ElectricalLoad | A class which contains time and electrical load data. Intended to serve as a component class of Model | 26 |
| Emissions | A structure which bundles the emitted masses of various emissions chemistries | 27 |
| Lilon | A derived class of Storage which models energy storage by way of lithium-ion batteries | 29 |
| 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 | 31 |
| Production | The base class of the Production hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise | 34 |
| ProductionInputs | A structure which bundles the necessary inputs for the Production constructor. Provides default values for every necessary input | 41 |
| Renewable | The root of the Renewable branch of the Production hierarchy. This branch contains derived classes which model the renewable production of energy | 43 |

Resources

A class which contains renewable resource data. Intended to serve as a component class of [Model](#) 45

Solar

A derived class of the [Renewable](#) branch of [Production](#) which models solar production 46

Storage

The base class of the [Storage](#) hierarchy. This hierarchy contains derived classes which model the storage of energy 48

Tidal

A derived class of the [Renewable](#) branch of [Production](#) which models tidal production 49

Wave

A derived class of the [Renewable](#) branch of [Production](#) which models wave production 51

Wind

A derived class of the [Renewable](#) branch of [Production](#) which models wind production 53

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

| | |
|--|----|
| header/ Controller.h | |
| Header file the Controller class | 57 |
| header/ ElectricalLoad.h | |
| Header file the ElectricalLoad class | 58 |
| header/ Model.h | |
| Header file the Model class | 59 |
| header/ Resources.h | |
| Header file the Resources class | 67 |
| header/ std_includes.h | |
| Header file which simply batches together the usual, standard includes | 68 |
| header/Production/ Production.h | |
| Header file the Production class | 62 |
| header/Production/Combustion/ Combustion.h | |
| Header file the Combustion class | 60 |
| header/Production/Combustion/ Diesel.h | |
| Header file the Diesel class | 61 |
| header/Production/Renewable/ Renewable.h | |
| Header file the Renewable class | 63 |
| header/Production/Renewable/ Solar.h | |
| Header file the Solar class | 63 |
| header/Production/Renewable/ Tidal.h | |
| Header file the Tidal class | 64 |
| header/Production/Renewable/ Wave.h | |
| Header file the Wave class | 65 |
| header/Production/Renewable/ Wind.h | |
| Header file the Wind class | 66 |
| header/Storage/ Lilon.h | |
| Header file the Lilon class | 69 |
| header/Storage/ Storage.h | |
| Header file the Storage class | 69 |
| pybindings/ PYBIND11_PGM.cpp | |
| Python 3 bindings file for PGMcpp | 70 |
| source/ Controller.cpp | |
| Implementation file for the Controller class | 72 |
| source/ ElectricalLoad.cpp | |
| Implementation file for the ElectricalLoad class | 73 |

| | |
|--|-----|
| source/ Model.cpp | |
| Implementation file for the Model class | 73 |
| source/ Resources.cpp | |
| Implementation file for the Resources class | 78 |
| source/Production/ Production.cpp | |
| Implementation file for the Production class | 75 |
| source/Production/Combustion/ Combustion.cpp | |
| Implementation file for the Combustion class | 74 |
| source/Production/Combustion/ Diesel.cpp | |
| Implementation file for the Diesel class | 74 |
| source/Production/Renewable/ Renewable.cpp | |
| Implementation file for the Renewable class | 75 |
| source/Production/Renewable/ Solar.cpp | |
| Implementation file for the Solar class | 76 |
| source/Production/Renewable/ Tidal.cpp | |
| Implementation file for the Tidal class | 76 |
| source/Production/Renewable/ Wave.cpp | |
| Implementation file for the Wave class | 77 |
| source/Production/Renewable/ Wind.cpp | |
| Implementation file for the Wind class | 77 |
| source/Storage/ Lilon.cpp | |
| Implementation file for the Lilon class | 78 |
| source/Storage/ Storage.cpp | |
| Implementation file for the Storage class | 79 |
| test/source/ test_Controller.cpp | |
| Testing suite for Controller class | 94 |
| test/source/ test_ElectricalLoad.cpp | |
| Testing suite for ElectricalLoad class | 95 |
| test/source/ test_Model.cpp | |
| Testing suite for Model class | 97 |
| test/source/ test_Resources.cpp | |
| Testing suite for Resources class | 98 |
| test/source/Production/ test_Production.cpp | |
| Testing suite for Production class | 90 |
| test/source/Production/Combustion/ test_Combustion.cpp | |
| Testing suite for Combustion class | 79 |
| test/source/Production/Combustion/ test_Diesel.cpp | |
| Testing suite for Diesel class | 81 |
| test/source/Production/Renewable/ test_Renewable.cpp | |
| Testing suite for Renewable class | 83 |
| test/source/Production/Renewable/ test_Solar.cpp | |
| Testing suite for Solar class | 85 |
| test/source/Production/Renewable/ test_Tidal.cpp | |
| Testing suite for Tidal class | 86 |
| test/source/Production/Renewable/ test_Wave.cpp | |
| Testing suite for Wave class | 87 |
| test/source/Production/Renewable/ test_Wind.cpp | |
| Testing suite for Wind class | 88 |
| test/source/Storage/ test_Lilon.cpp | |
| Testing suite for Lilon class | 92 |
| test/source/Storage/ test_Storage.cpp | |
| Testing suite for Storage class | 93 |
| test/utills/ testing_utils.cpp | |
| Header file for various PGMcpp testing utilities | 99 |
| test/utills/ testing_utils.h | |
| Header file for various PGMcpp testing utilities | 105 |

Chapter 4

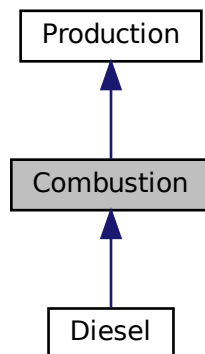
Class Documentation

4.1 Combustion Class Reference

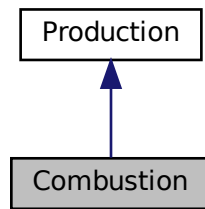
The root of the [Combustion](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

```
#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.

```
55 {
56     return;
57 } /* Combustion() */
```

4.1.2.2 Combustion() [2/2]

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

Constructor (intended) for the [Combustion](#) class.

Parameters

| | |
|--------------------------|---|
| <i>n_points</i> | The number of points in the modelling time series. |
| <i>combustion_inputs</i> | A structure of Combustion constructor inputs. |

```
75
76 Production(n_points, combustion_inputs.production_inputs)
77 {
78     // 1. check inputs
79     this->__checkInputs(combustion_inputs);
80
81     // 2. set attributes
82     this->linear_fuel_slope_LkWh = 0;
83     this->linear_fuel_intercept_LkWh = 0;
84
85     this->fuel_consumption_vec_L.resize(this->n_points, 0);
86     this->fuel_cost_vec.resize(this->n_points, 0);
87 }
```

```

88     this->CO2_emissions_vec_kg.resize(this->n_points, 0);
89     this->CO_emissions_vec_kg.resize(this->n_points, 0);
90     this->NOx_emissions_vec_kg.resize(this->n_points, 0);
91     this->SOx_emissions_vec_kg.resize(this->n_points, 0);
92     this->CH4_emissions_vec_kg.resize(this->n_points, 0);
93     this->PM_emissions_vec_kg.resize(this->n_points, 0);
94
95     // 3. construction print
96     if (this->print_flag) {
97         std::cout << "Combustion object constructed at " << this << std::endl;
98     }
99
100    return;
101 } /* Combustion() */

```

4.1.2.3 ~Combustion()

```

Combustion::~Combustion (
    void ) [virtual]

```

Destructor for the [Combustion](#) class.

```

226 {
227     // 1. destruction print
228     if (this->print_flag) {
229         std::cout << "Combustion object at " << this << " destroyed" << std::endl;
230     }
231
232     return;
233 } /* ~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

| | |
|----------------------|--|
| <i>timestep</i> | The timestep (i.e., time series index) for the request. |
| <i>dt_hrs</i> | The interval of time [hrs] associated with the timestep. |
| <i>production_kW</i> | The production [kW] of the asset in this timestep. |
| <i>load_kW</i> | 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](#).

```

137 {
138     // 1. invoke base class method
139     load_kW = Production :: commit(
140         timestep,
141         dt_hrs,
142         production_kW,
143         load_kW
144     );
145
146
147     if (this->is_running) {
148         // 2. compute and record fuel consumption
149         double fuel_consumed_L = this->getFuelConsumptionL(dt_hrs, production_kW);
150         this->fuel_consumption_vec_L[timestep] = fuel_consumed_L;
151
152         // 3. compute and record emissions
153         //...
154
155         // 4. incur fuel costs
156         //...
157     }
158
159     return load_kW;
160 }

```

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

| | |
|------------------------|----------------------------------|
| <i>fuel_consumed_L</i> | The volume of fuel consumed [L]. |
|------------------------|----------------------------------|

Returns

[Emissions](#) A structure containing the mass spectrum of resulting emissions.

```

205                                     {
206     Emissions emissions;
207
208     //...
209
210     return emissions;
211 } /* 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

| | |
|----------------------|--|
| <i>dt_hrs</i> | The interval of time [hrs] associated with the timestep. |
| <i>production_kW</i> | The production [kW] of the asset in this timestep. |

```

180 {
181     double fuel_consumed_L = (
182         this->linear_fuel_slope_LkWh * production_kW +
183         this->linear_fuel_intercept_LkWh * this->capacity_kW
184     ) * dt_hrs;
185
186     return fuel_consumed_L;
187 } /* getFuelConsumption() */

```

4.1.3.4 requestProductionkW()

```

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

```

Reimplemented in [Diesel](#).

```

106 {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:

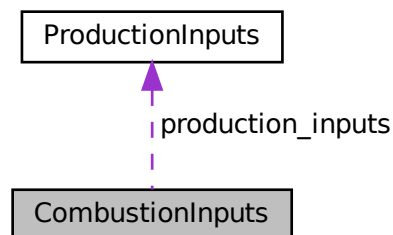
- [header/Production/Combustion/Combustion.h](#)
- [source/Production/Combustion/Combustion.cpp](#)

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.

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

4.3.2.2 ~Controller()

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

Destructor for the [Controller](#) class.

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

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

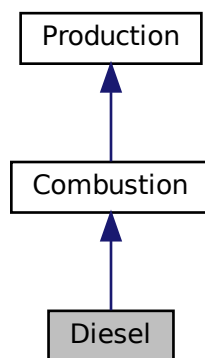
- header/[Controller.h](#)
- source/[Controller.cpp](#)

4.4 Diesel Class Reference

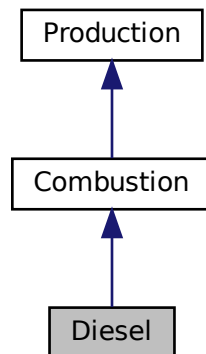
A derived class of the [Combustion](#) branch of [Production](#) which models production using a diesel generator.

```
#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

| | |
|----------------------|---|
| <i>n_points</i> | The number of points in the modelling time series. |
| <i>diesel_inputs</i> | A structure of Diesel constructor inputs. |

```
260 {
261     return;
262 } /* Diesel() */
```

4.4.2.2 Diesel() [2/2]

```
Diesel::Diesel (
    int n_points,
    DieselInputs diesel_inputs )
280 :
281 Combustion(n_points, diesel_inputs.combustion_inputs)
282 {
283     // 1. check inputs
284     this->__checkInputs(diesel_inputs);
285
286     // 2. set attributes
287     this->fuel_cost_L = diesel_inputs.fuel_cost_L;
288
289     this->minimum_load_ratio = diesel_inputs.minimum_load_ratio;
290     this->minimum_runtime_hrs = diesel_inputs.minimum_runtime_hrs;
291     this->time_since_last_start_hrs = 0;
292 }
```

```

293     this->CO2_emissions_intensity_kgL = diesel_inputs.CO2_emissions_intensity_kgL;
294     this->CO_emissions_intensity_kgL = diesel_inputs.CO_emissions_intensity_kgL;
295     this->NOx_emissions_intensity_kgL = diesel_inputs.NOx_emissions_intensity_kgL;
296     this->SOx_emissions_intensity_kgL = diesel_inputs.SOx_emissions_intensity_kgL;
297     this->CH4_emissions_intensity_kgL = diesel_inputs.CH4_emissions_intensity_kgL;
298     this->PM_emissions_intensity_kgL = diesel_inputs.PM_emissions_intensity_kgL;
299
300     if (diesel_inputs.linear_fuel_slope_LkWh < 0) {
301         this->linear_fuel_slope_LkWh = this->__getGenericFuelSlope();
302     }
303
304
305     if (diesel_inputs.linear_fuel_intercept_LkWh < 0) {
306         this->linear_fuel_intercept_LkWh = this->__getGenericFuelIntercept();
307     }
308
309     if (diesel_inputs.capital_cost < 0) {
310         this->capital_cost = this->__getGenericCapitalCost();
311     }
312
313     if (diesel_inputs.operation_maintenance_cost_kWh < 0) {
314         this->operation_maintenance_cost_kWh = this->__getGenericOpMaintCost();
315     }
316
317     if (this->is_sunk) {
318         this->capital_cost_vec[0] = this->capital_cost;
319     }
320
321     // 3. construction print
322     if (this->print_flag) {
323         std::cout << "Diesel object constructed at " << this << std::endl;
324     }
325
326     return;
327 } /* Diesel() */

```

4.4.2.3 ~Diesel()

```

Diesel::~Diesel (
    void )

```

Destructor for the [Diesel](#) class.

```

438 {
439     // 1. destruction print
440     if (this->print_flag) {
441         std::cout << "Diesel object at " << this << " destroyed" << std::endl;
442     }
443
444     return;
445 } /* ~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

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

Reimplemented from [Combustion](#).

```

405 {
406     // 1. handle start/stop, enforce minimum runtime constraint
407     this->__handleStartStop(timestep, dt_hrs, production_kW);
408
409     // 2. invoke base class method
410     load_kW = Combustion::commit(
411         timestep,
412         dt_hrs,
413         production_kW,
414         load_kW
415     );
416
417     if (this->is_running) {
418         // 3. log time since last start
419         this->time_since_last_start_hrs += dt_hrs;
420     }
421
422     return load_kW;
423 } /* 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

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

Reimplemented from [Combustion](#).

```

357 {
358     double deliver_kW = request_kW;
359
360     // 1. enforce capacity constraint
361     if (deliver_kW > this->capacity_kW) {
362         deliver_kW = this->capacity_kW;
363     }
364
365     // 2. enforce minimum load ratio
366     if (deliver_kW < this->minimum_load_ratio * this->capacity_kW) {
367         deliver_kW = this->minimum_load_ratio * this->capacity_kW;
368     }
369
370     return deliver_kW;
371 } /* 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:

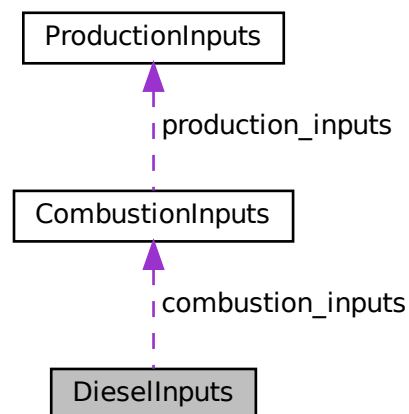
- [header/Production/Combustion/Diesel.h](#)
- [source/Production/Combustion/Diesel.cpp](#)

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.

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

4.6.2.2 ~ElectricalLoad()

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

Destructor for the [ElectricalLoad](#) class.

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

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

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

4.7 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:

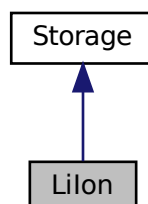
- [header/Production/Combustion/Combustion.h](#)

4.8 Lilon Class Reference

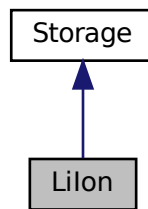
A derived class of [Storage](#) which models energy storage by way of lithium-ion batteries.

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

Inheritance diagram for Lilon:



Collaboration diagram for Lilon:



Public Member Functions

- `Lilon` (void)
Constructor for the `Lilon` class.
- `~Lilon` (void)
Destructor for the `Lilon` class.

4.8.1 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 `Lilon()`

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

Constructor for the `Lilon` class.

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


4.8.2.2 ~Lilon()

```

Lilon::~Lilon (
    void )

```

Destructor for the [Lilon](#) class.

```

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

```

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

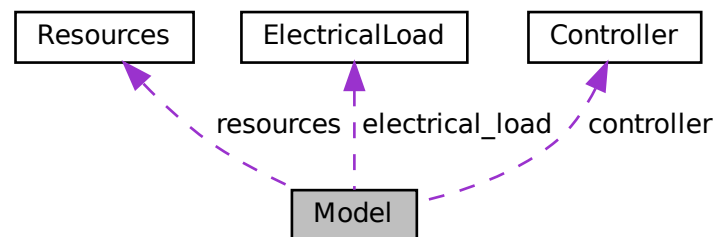
- header/Storage/[Lilon.h](#)
- source/Storage/[Lilon.cpp](#)

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.

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

4.9.2.2 ~Model()

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

Destructor for the [Model](#) class.

```
64 {
65     //...
66
67     return;
68 } /* ~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:

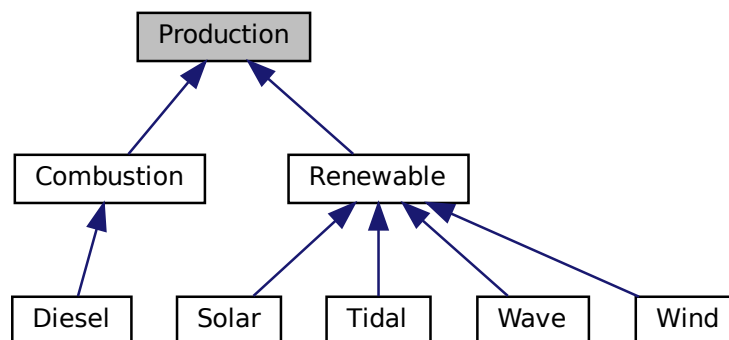
- header/[Model.h](#)
- source/[Model.cpp](#)

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 asset 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.

```
96 {
97     return;
98 } /* Production() */
```

4.10.2.2 Production() [2/2]

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

Constructor (intended) for the [Production](#) class.

Parameters

| | |
|--------------------------|---|
| <i>n_points</i> | The number of points in the modelling time series. |
| <i>production_inputs</i> | A structure of Production constructor inputs. |

```
120 {
121     // 1. check inputs
122     this->__checkInputs(n_points, production_inputs);
123
124     // 2. set attributes
125     this->print_flag = production_inputs.print_flag;
126     this->is_running = false;
127
128     this->n_points = n_points;
129     this->n_starts = 0;
130
131     this->running_hours = 0;
132
133     this->capacity_kW = production_inputs.capacity_kW;
134
135     this->real_discount_annual = this->__computeRealDiscountAnnual(
136         production_inputs.nominal_inflation_annual,
137         production_inputs.nominal_discount_annual
138     );
139     this->capital_cost = 0;
140     this->operation_maintenance_cost_kWh = 0;
141     this->net_present_cost = 0;
142     this->levellized_cost_of_energy_kWh = 0;
143
144     this->production_vec_kW.resize(this->n_points, 0);
145     this->dispatch_vec_kW.resize(this->n_points, 0);
146     this->storage_vec_kW.resize(this->n_points, 0);
147     this->curtailment_vec_kW.resize(this->n_points, 0);
148
149     this->capital_cost_vec.resize(this->n_points, 0);
150     this->operation_maintenance_cost_vec.resize(this->n_points, 0);
151
152     // 3. construction print
153     if (this->print_flag) {
154         std::cout << "Production object constructed at " << this << std::endl;
155     }
156
157     return;
```

```
158 }    /* Production() */
```

4.10.2.3 ~Production()

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

Destructor for the [Production](#) class.

```
243 {
244     // 1. destruction print
245     if (this->print_flag) {
246         std::cout << "Production object at " << this << " destroyed" << std::endl;
247     }
248
249     return;
250 }    /* ~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

| | |
|----------------------|--|
| <i>timestep</i> | The timestep (i.e., time series index) for the request. |
| <i>dt_hrs</i> | The interval of time [hrs] associated with the timestep. |
| <i>production_kW</i> | The production [kW] of the asset in this timestep. |
| <i>load_kW</i> | 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](#).

```
194 {
195     // 1. record production
196     this->production_vec_kW[timestep] = production_kW;
197
198     // 2. compute and record dispatch and curtailment
199     double dispatch_kW = 0;
200     double curtailment_kW = 0;
201
202     if (production_kW > load_kW) {
```

```

203         dispatch_kW = load_kW;
204         curtailment_kW = production_kW - dispatch_kW;
205     }
206
207     else {
208         dispatch_kW = production_kW;
209     }
210
211     this->dispatch_vec_kW[timestep] = dispatch_kW;
212     this->curtailment_vec_kW[timestep] = curtailment_kW;
213
214     // 3. update load
215     load_kW -= dispatch_kW;
216
217     if (this->is_running) {
218         // 4. log running state, running hours
219         this->is_running_vec[timestep] = this->is_running;
220         this->running_hours += dt_hrs;
221
222         // 5. incur capital and operating costs
223         //...
224     }
225
226
227     return load_kW;
228 }

```

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 asset 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:

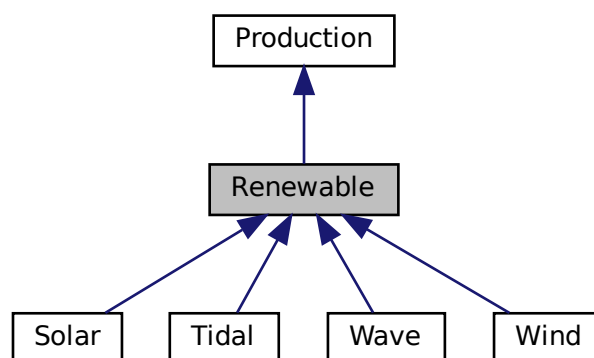
- [header/Production/Production.h](#)

4.12 Renewable Class Reference

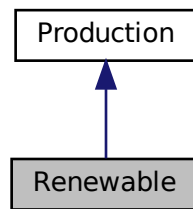
The root of the [Renewable](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the renewable production of energy.

```
#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.

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

4.12.2.2 ~Renewable()

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

Destructor for the [Renewable](#) class.

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

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

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

4.13 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.

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

4.13.2.2 ~Resources()

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

Destructor for the [Resources](#) class.

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

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

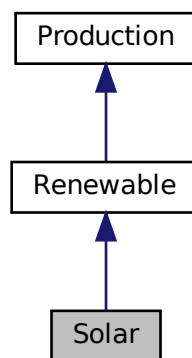
- header/[Resources.h](#)
- source/[Resources.cpp](#)

4.14 Solar Class Reference

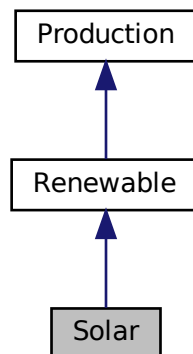
A derived class of the [Renewable](#) branch of [Production](#) which models solar production.

```
#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.

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

4.14.2.2 ~Solar()

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

Destructor for the [Solar](#) class.

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

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

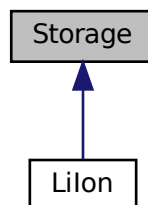
- [header/Production/Renewable/Solar.h](#)
- [source/Production/Renewable/Solar.cpp](#)

4.15 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.

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

4.15.2.2 ~Storage()

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

Destructor for the [Storage](#) class.

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

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

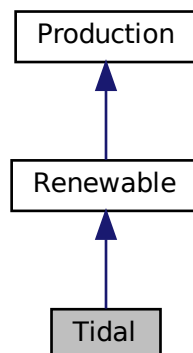
- [header/Storage/Storage.h](#)
- [source/Storage/Storage.cpp](#)

4.16 Tidal Class Reference

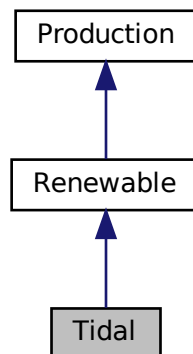
A derived class of the [Renewable](#) branch of [Production](#) which models tidal production.

```
#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.

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

4.16.2.2 ~Tidal()

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

Destructor for the [Tidal](#) class.

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

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

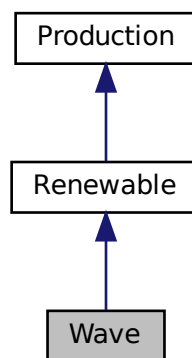
- [header/Production/Renewable/Tidal.h](#)
- [source/Production/Renewable/Tidal.cpp](#)

4.17 Wave Class Reference

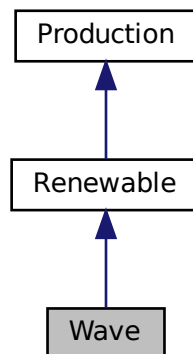
A derived class of the [Renewable](#) branch of [Production](#) which models wave production.

```
#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.

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

4.17.2.2 ~Wave()

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

Destructor for the [Wave](#) class.

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

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

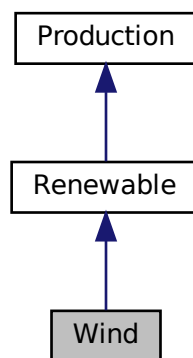
- [header/Production/Renewable/Wave.h](#)
- [source/Production/Renewable/Wave.cpp](#)

4.18 Wind Class Reference

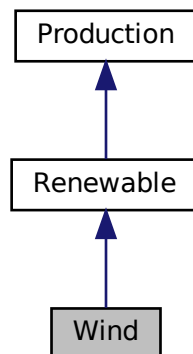
A derived class of the [Renewable](#) branch of [Production](#) which models wind production.

```
#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.

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


4.18.2.2 ~Wind()

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

Destructor for the [Wind](#) class.

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

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

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

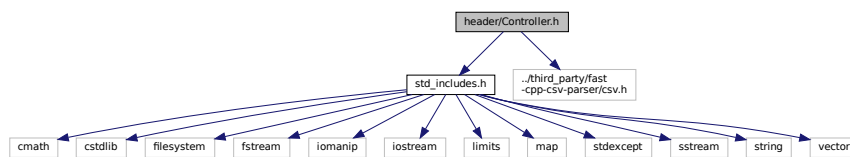
Chapter 5

File Documentation

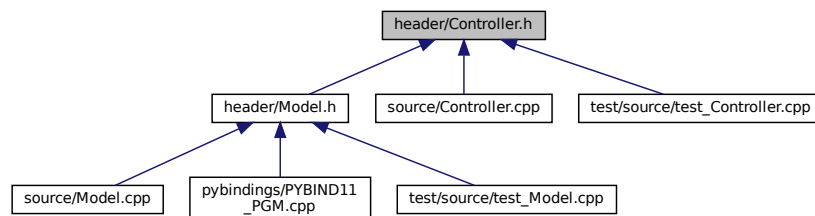
5.1 header/Controller.h File Reference

Header file the [Controller](#) class.

```
#include "std_includes.h"
#include "../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](#).

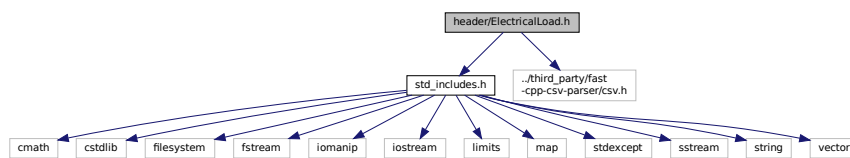
5.1.1 Detailed Description

Header file the [Controller](#) class.

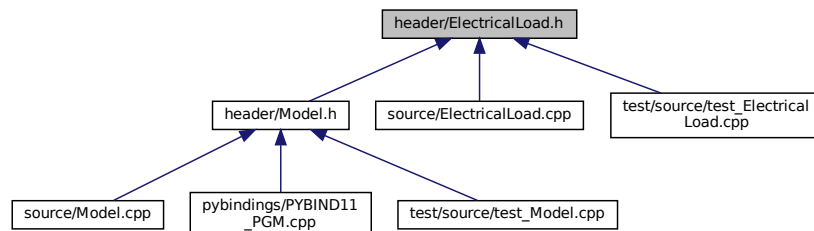
5.2 header/ElectricalLoad.h File Reference

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

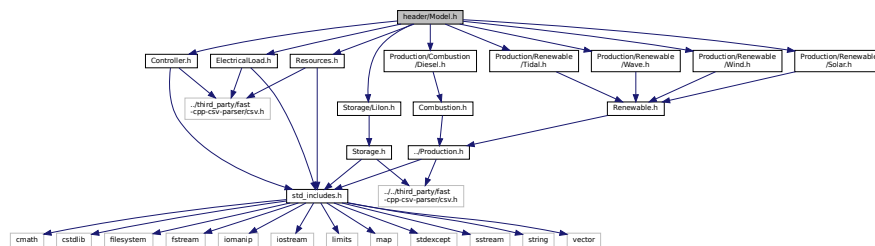
Header file the [ElectricalLoad](#) class.

5.3 header/Model.h File Reference

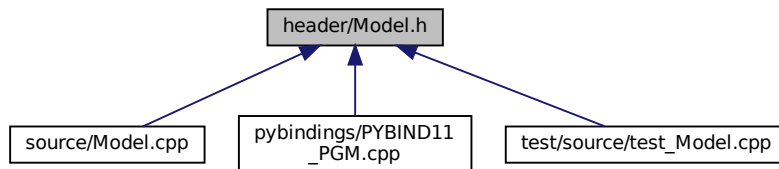
Header file the [Model](#) class.

```
#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:



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



Classes

- class [Model](#)

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

5.3.1 Detailed Description

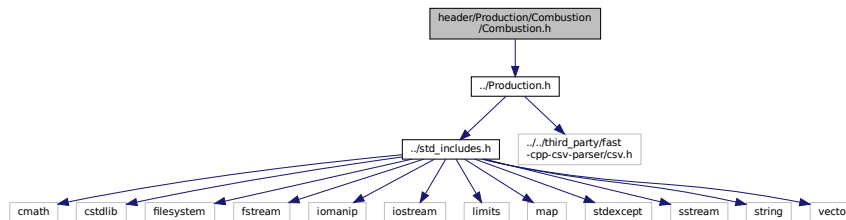
Header file the [Model](#) class.

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

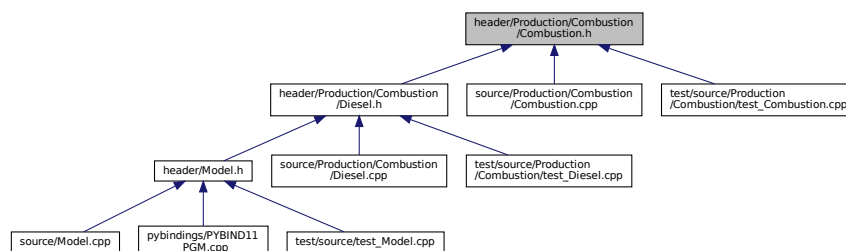
Header file the [Combustion](#) class.

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

Include dependency graph for Combustion.h:



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



Classes

- 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. |

```

33         {
34     DIESEL,
35     N_COMBUSTION_TYPES
36 };

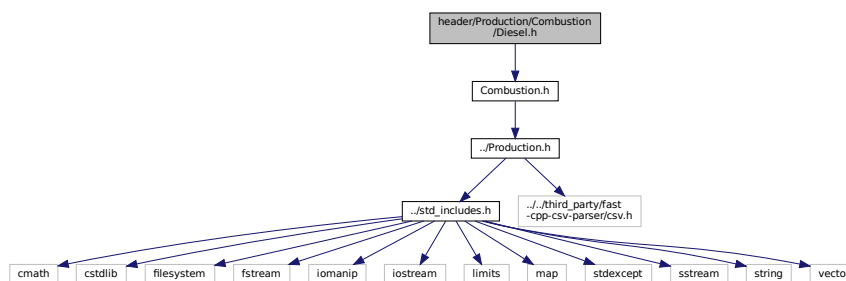
```

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

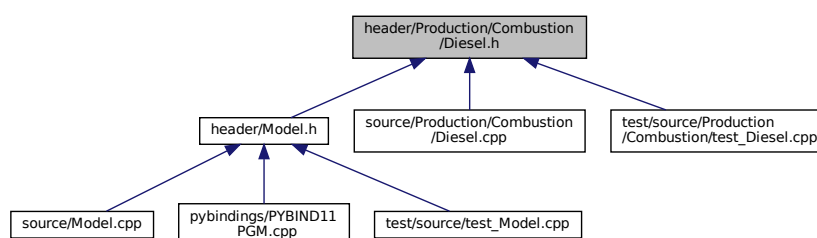
Header file the `Diesel` class.

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

Include dependency graph for Diesel.h:



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

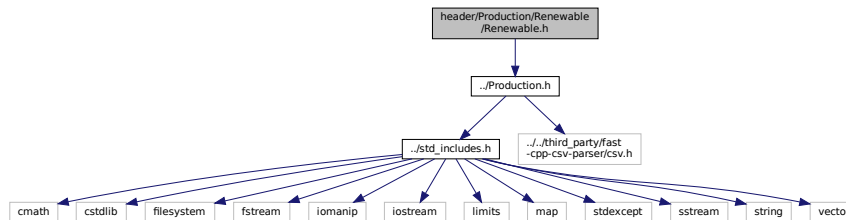


5.7 header/Production/Renewable/Renewable.h File Reference

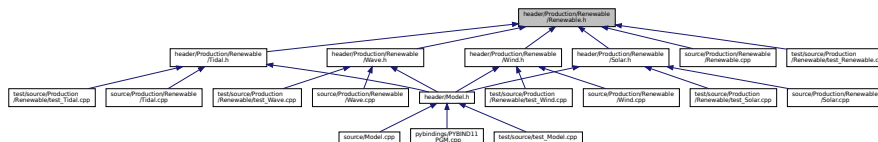
Header file the [Renewable](#) class.

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

Include dependency graph for Renewable.h:



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



Classes

- class [Renewable](#)

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

5.7.1 Detailed Description

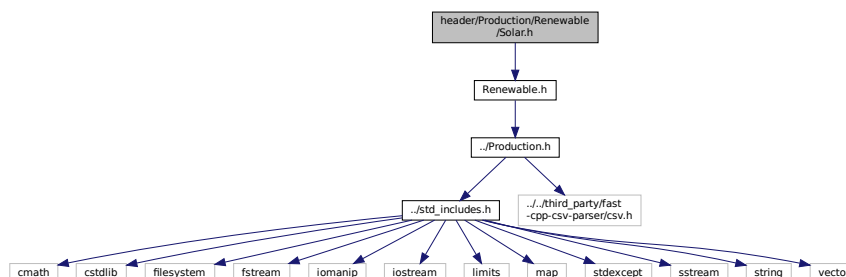
Header file the [Renewable](#) class.

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

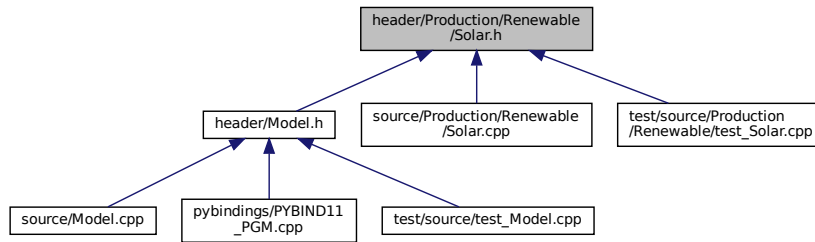
Header file the [Solar](#) class.

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

Include dependency graph for Solar.h:



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



Classes

- class [Solar](#)

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

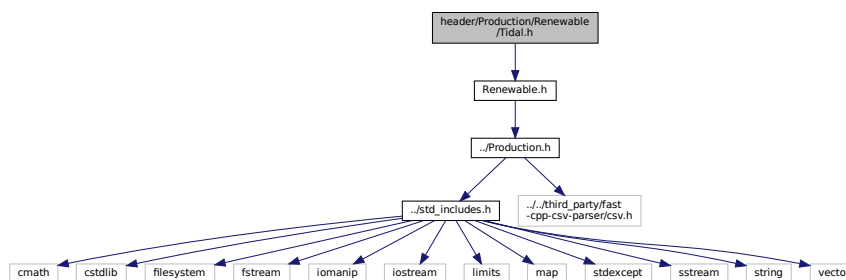
5.8.1 Detailed Description

Header file the [Solar](#) class.

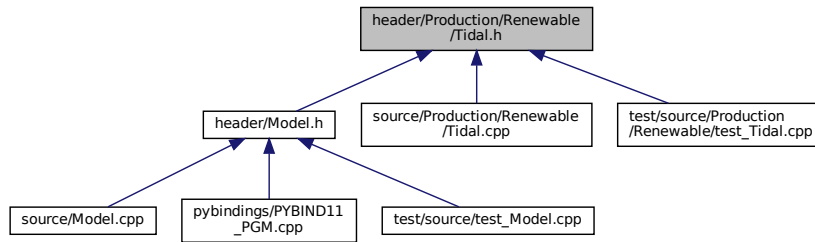
5.9 header/Production/Renewable/Tidal.h File Reference

Header file the [Tidal](#) class.

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



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



Classes

- class [Tidal](#)

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

5.9.1 Detailed Description

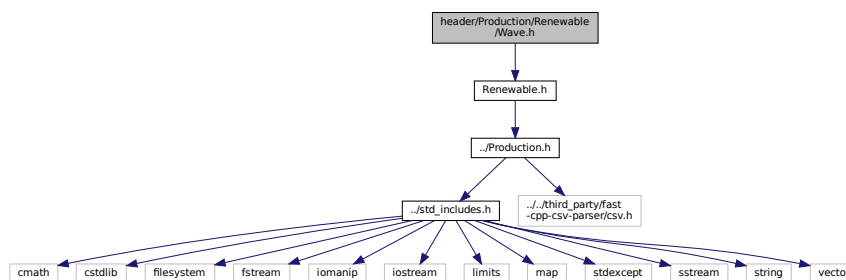
Header file the [Tidal](#) class.

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

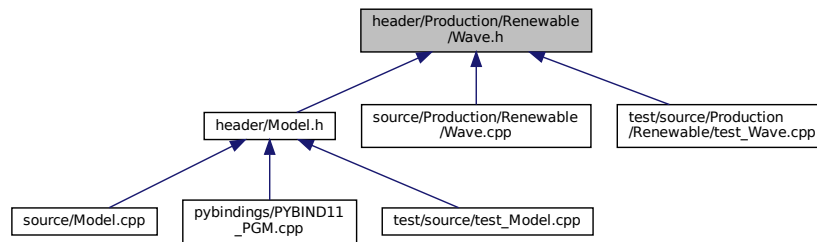
Header file the [Wave](#) class.

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

Include dependency graph for Wave.h:



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



Classes

- class [Wave](#)

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

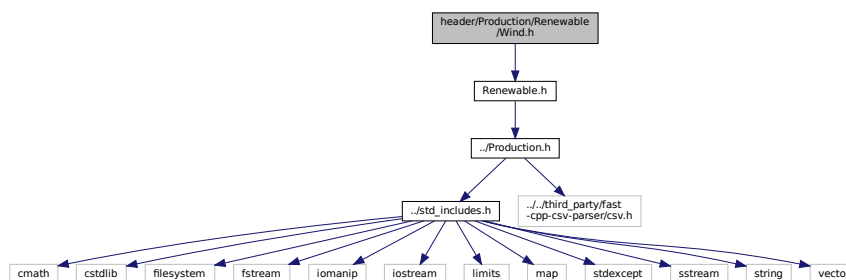
5.10.1 Detailed Description

Header file the [Wave](#) class.

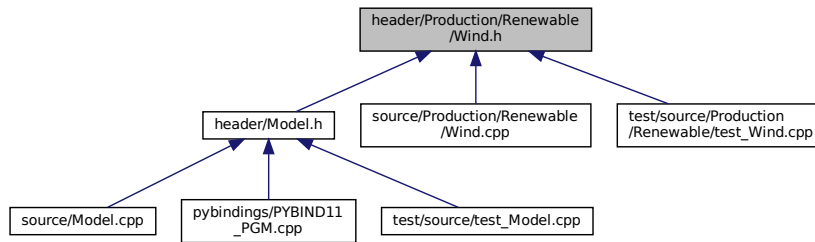
5.11 header/Production/Renewable/Wind.h File Reference

Header file the [Wind](#) class.

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



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



Classes

- class [Wind](#)

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

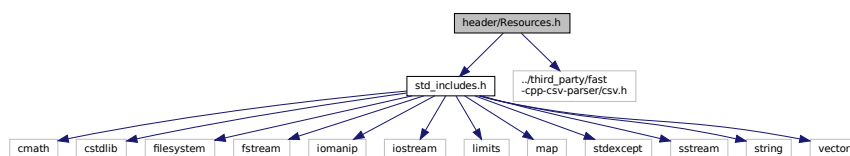
5.11.1 Detailed Description

Header file the [Wind](#) class.

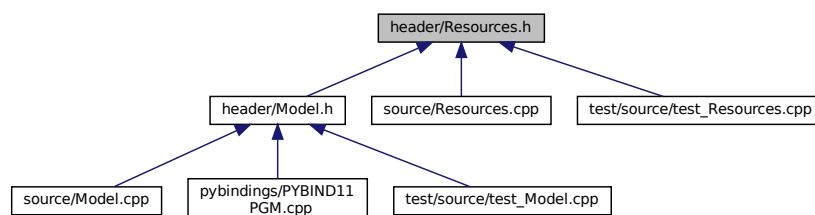
5.12 header/Resources.h File Reference

Header file the [Resources](#) class.

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

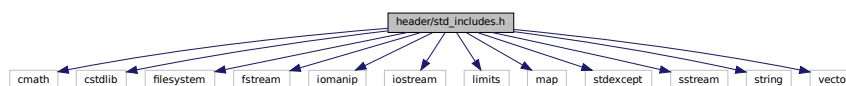
Header file the [Resources](#) class.

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

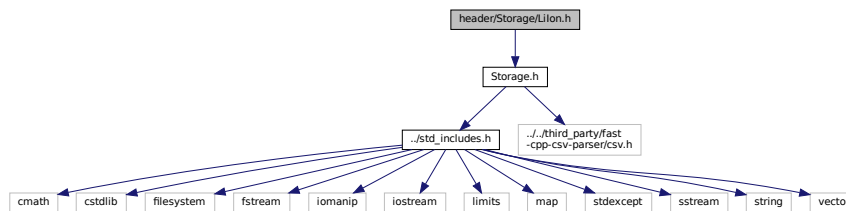
Header file which simply batches together the usual, standard includes.

5.14 header/Storage/Lilon.h File Reference

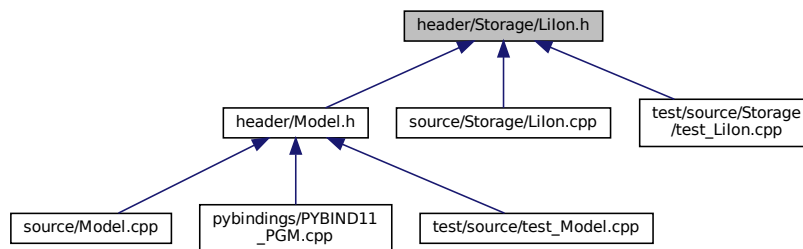
Header file the [Lilon](#) class.

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

Include dependency graph for Lilon.h:



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



Classes

- class [Lilon](#)

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

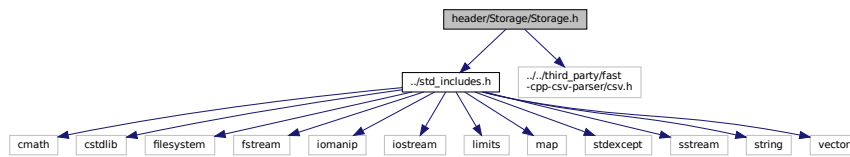
5.14.1 Detailed Description

Header file the [Lilon](#) class.

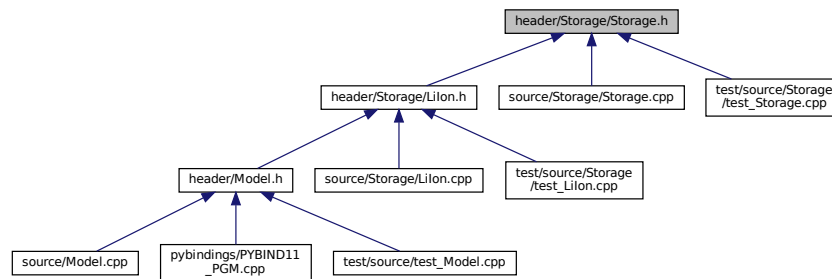
5.15 header/Storage/Storage.h File Reference

Header file the [Storage](#) class.

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

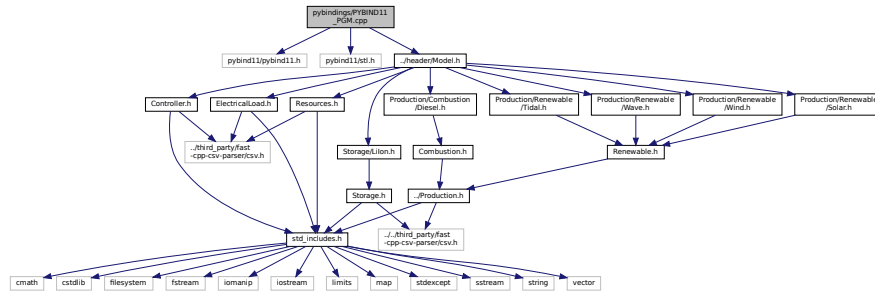
Python 3 bindings file for PGMcpp.

```
#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 )
{
30
31
32 // ===== Controller ===== //
33 /*
34 pybind11::class_<Controller>(m, "Controller")
35     .def(pybind11::init());
36 */
37 // ===== END Controller ===== //
38
39
40
41 // ===== ElectricalLoad ===== //
42 /*
43 pybind11::class_<ElectricalLoad>(m, "ElectricalLoad")
44     .def_readwrite("n_points", &ElectricalLoad::n_points)
45     .def_readwrite("max_load_kW", &ElectricalLoad::max_load_kW)
46     .def_readwrite("mean_load_kW", &ElectricalLoad::mean_load_kW)
47     .def_readwrite("min_load_kW", &ElectricalLoad::min_load_kW)
48     .def_readwrite("dt_vec_hrs", &ElectricalLoad::dt_vec_hrs)
49     .def_readwrite("load_vec_kW", &ElectricalLoad::load_vec_kW)
50     .def_readwrite("time_vec_hrs", &ElectricalLoad::time_vec_hrs)
```

```

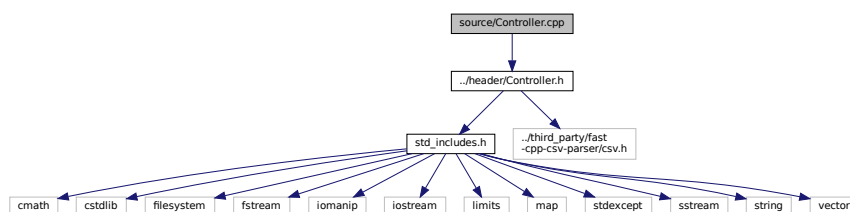
51
52     .def(pybind11::init<std::string>());
53 */
54 // ===== END ElectricalLoad ===== //
55
56
57
58 // ===== Model ===== //
59 /*
60 pybind11::class_<Model>(m, "Model")
61     .def(
62         pybind11::init<
63             ElectricalLoad*,
64             RenewableResources*
65         >()
66     );
67 */
68 // ===== END Model ===== //
69
70
71
72 // ===== RenewableResources ===== //
73 /*
74 pybind11::class_<RenewableResources>(m, "RenewableResources")
75     .def(pybind11::init());
76     /*
77     .def(pybind11::init<>());
78     */
79 */
80 // ===== END RenewableResources ===== //
81
82 } /* PYBIND11_MODULE() */

```

5.17 source/Controller.cpp File Reference

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

#include "../header/Controller.h"
 Include dependency graph for Controller.cpp:



5.17.1 Detailed Description

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

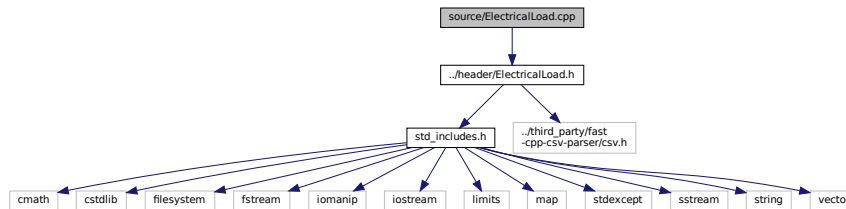
A class which contains a various dispatch control logic. Intended to serve as a component class of [Controller](#).

5.18 source/ElectricalLoad.cpp File Reference

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

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

Include dependency graph for ElectricalLoad.cpp:



5.18.1 Detailed Description

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

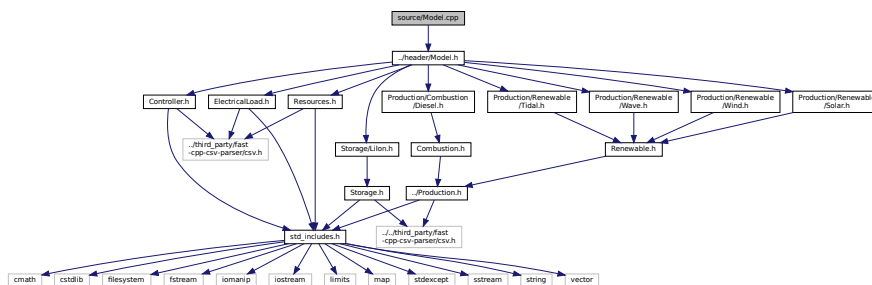
A class which contains time and electrical load data. Intended to serve as a component class of [Model](#).

5.19 source/Model.cpp File Reference

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

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

Include dependency graph for Model.cpp:



5.19.1 Detailed Description

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

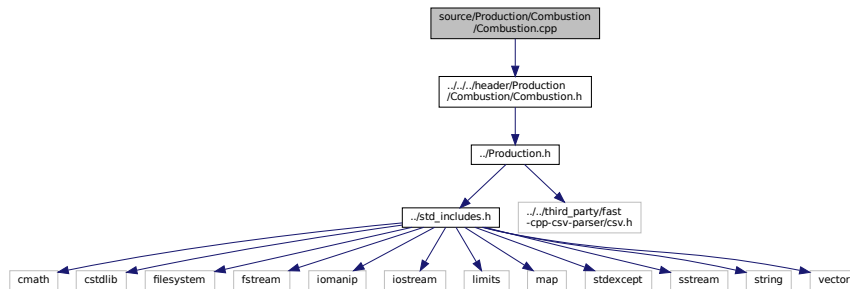
A container class which forms the centre of PGMcpp. The [Model](#) class is intended to serve as the primary user interface with the functionality of PGMcpp, and as such it contains all other classes.

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

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

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

Include dependency graph for Combustion.cpp:



5.20.1 Detailed Description

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

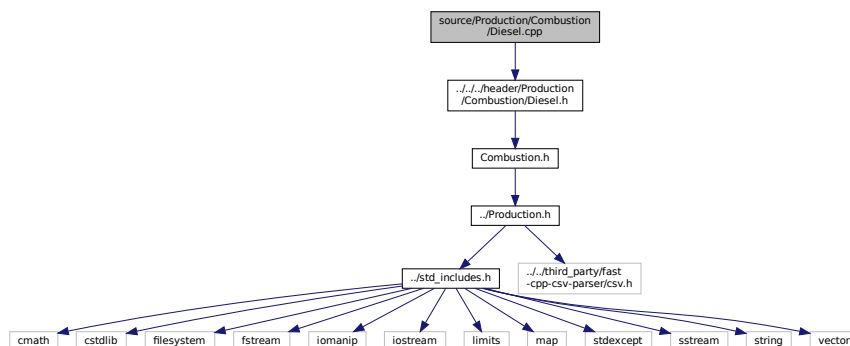
The root of the [Combustion](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the production of energy by way of combustibles.

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

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

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

Include dependency graph for Diesel.cpp:



5.21.1 Detailed Description

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

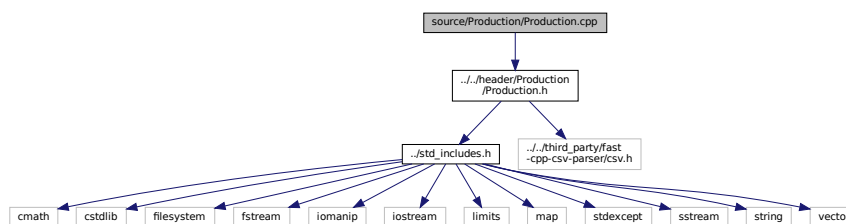
A derived class of the [Combustion](#) branch of [Production](#) which models production using a diesel generator.

5.22 source/Production/Production.cpp File Reference

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

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

Include dependency graph for Production.cpp:



5.22.1 Detailed Description

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

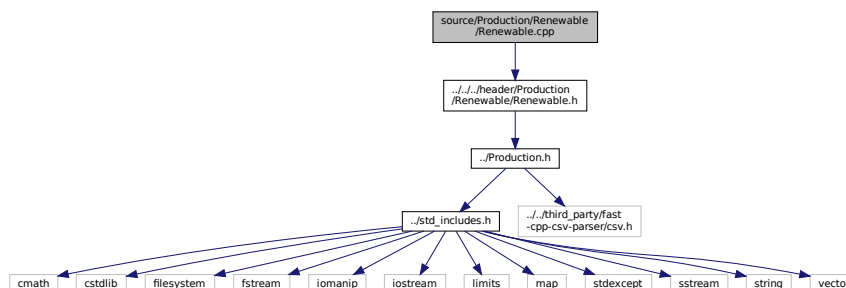
The base class of the [Production](#) hierarchy. This hierarchy contains derived classes which model the production of energy, be it renewable or otherwise.

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

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

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

Include dependency graph for Renewable.cpp:



5.23.1 Detailed Description

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

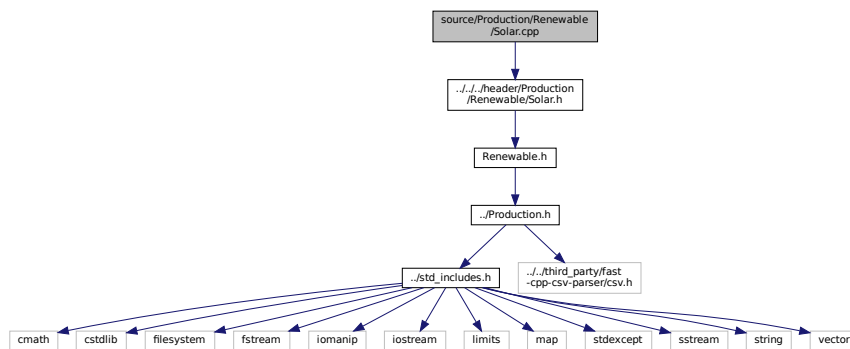
The root of the [Renewable](#) branch of the [Production](#) hierarchy. This branch contains derived classes which model the renewable production of energy.

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

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

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

Include dependency graph for Solar.cpp:



5.24.1 Detailed Description

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

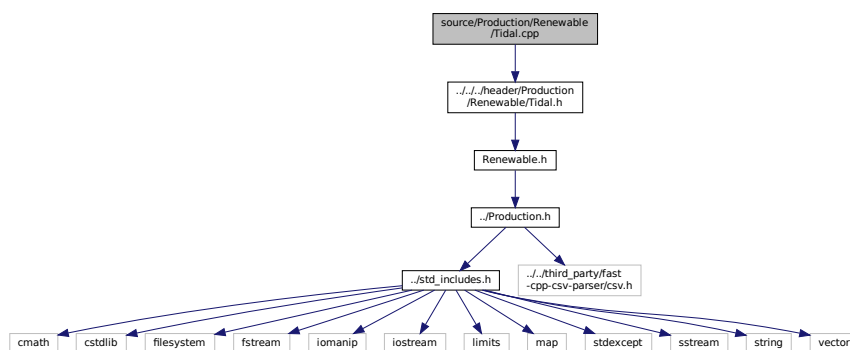
A derived class of the [Renewable](#) branch of [Production](#) which models solar production.

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

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

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

Include dependency graph for Tidal.cpp:



5.25.1 Detailed Description

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

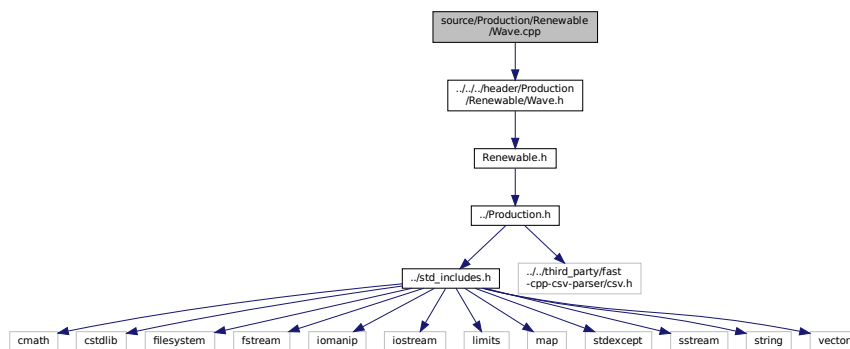
A derived class of the [Renewable](#) branch of [Production](#) which models tidal production.

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

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

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

Include dependency graph for Wave.cpp:



5.26.1 Detailed Description

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

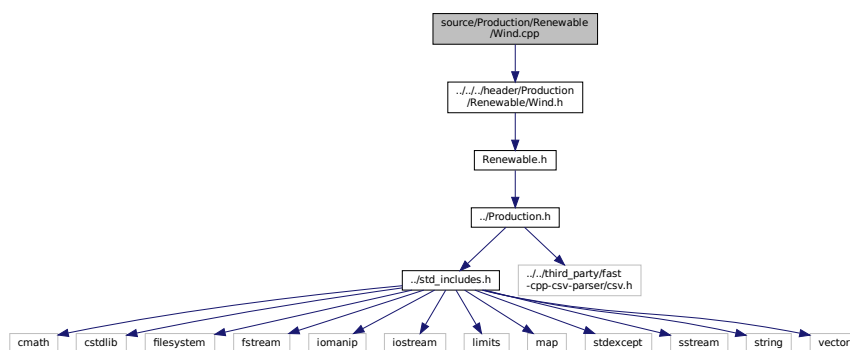
A derived class of the [Renewable](#) branch of [Production](#) which models wave production.

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

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

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

Include dependency graph for Wind.cpp:



5.27.1 Detailed Description

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

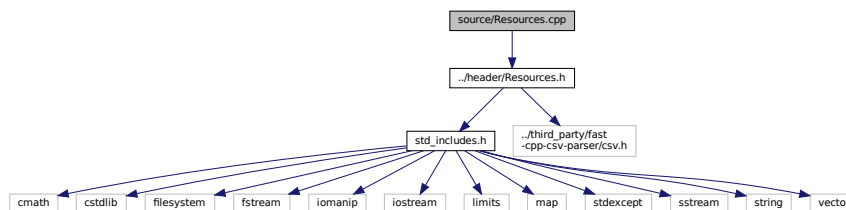
A derived class of the [Renewable](#) branch of [Production](#) which models wind production.

5.28 source/Resources.cpp File Reference

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

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

Include dependency graph for Resources.cpp:



5.28.1 Detailed Description

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

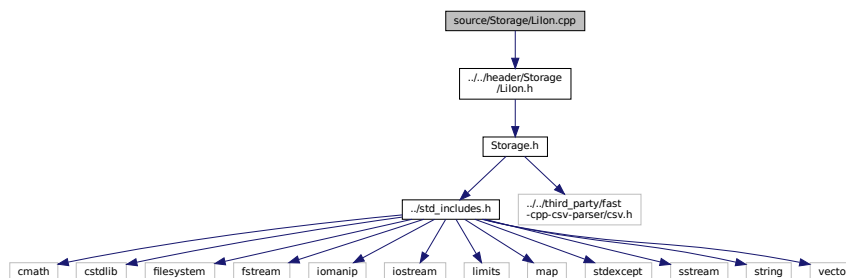
A class which contains renewable resource data. Intended to serve as a component class of [Model](#).

5.29 source/Storage/Lilon.cpp File Reference

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

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

Include dependency graph for Lilon.cpp:



5.29.1 Detailed Description

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

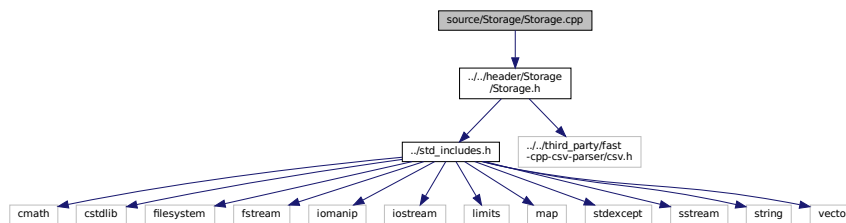
A derived class of [Storage](#) which models energy storage by way of lithium-ion batteries.

5.30 source/Storage/Storage.cpp File Reference

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

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

Include dependency graph for Storage.cpp:



5.30.1 Detailed Description

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

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

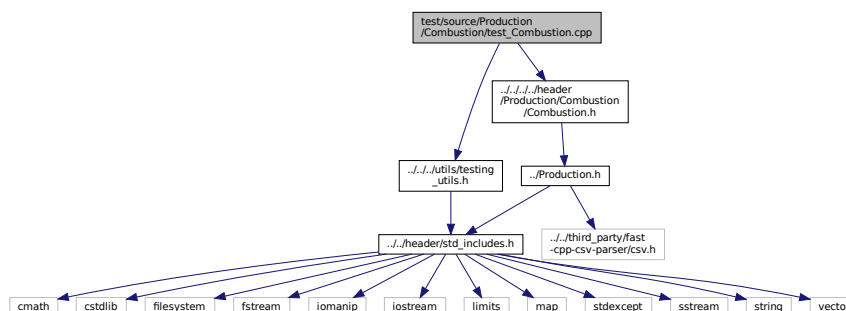
5.31 test/source/Production/Combustion/test_Combustion.cpp File Reference

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 )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Production <-- Combustion");
33
34     srand(time(NULL));
35
36
37     try {
38         // 1. construction
39         CombustionInputs combustion_inputs;
40
41         Combustion test_combustion(8760, combustion_inputs);
42
43
44         // 2. test structure attributes
45         testTruth(
46             not combustion_inputs.production_inputs.print_flag,
47             __FILE__,
48             __LINE__
49         );
50
51
52         // 3. test post-construction attributes
53         testFloatEquals(
54             test_combustion.fuel_consumption_vec_L.size(),
55             8760,
56             __FILE__,
57             __LINE__
58         );
59
60         testFloatEquals(
61             test_combustion.fuel_cost_vec.size(),
62             8760,
63             __FILE__,
64             __LINE__
65         );
66
67         testFloatEquals(
68             test_combustion.CO2_emissions_vec_kg.size(),
69             8760,
70             __FILE__,
71             __LINE__
72         );
73
74         testFloatEquals(
75             test_combustion.CO_emissions_vec_kg.size(),
76             8760,
77             __FILE__,
```

```

78     __LINE__
79 );
80
81 testFloatEquals(
82     test_combustion.NOx_emissions_vec_kg.size(),
83     8760,
84     __FILE__,
85     __LINE__
86 );
87
88 testFloatEquals(
89     test_combustion.SOx_emissions_vec_kg.size(),
90     8760,
91     __FILE__,
92     __LINE__
93 );
94
95 testFloatEquals(
96     test_combustion.CH4_emissions_vec_kg.size(),
97     8760,
98     __FILE__,
99     __LINE__
100 );
101
102 testFloatEquals(
103     test_combustion.PM_emissions_vec_kg.size(),
104     8760,
105     __FILE__,
106     __LINE__
107 );
108 }
109
110 catch (...) {
111     //...
112
113     printGold(" ..... ");
114     printRed("FAIL");
115     std::cout << std::endl;
116     throw;
117 }
118
119
120 printGold(" ..... ");
121 printGreen("PASS");
122 std::cout << std::endl;
123 return 0;
124 } /* main() */

```

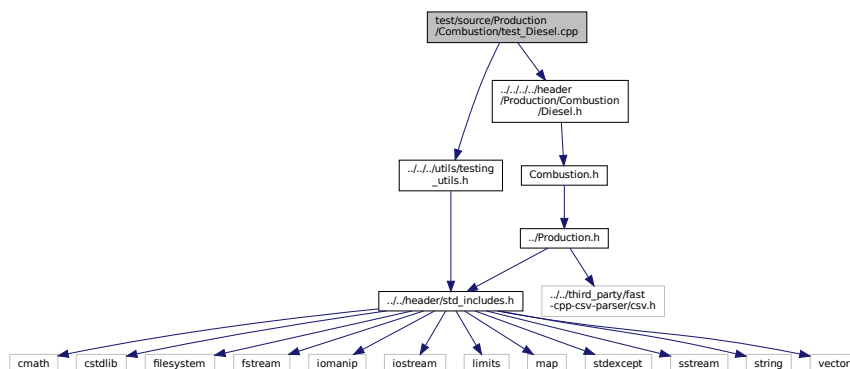
5.32 test/source/Production/Combustion/test_Diesel.cpp File Reference

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 )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Production <-- Combustion <-- Diesel");
33
34     srand(time(NULL));
35
36
37     try {
38         // 1. construction
39         bool error_flag = true;
40         try {
41             DieselInputs bad_diesel_inputs;
42             bad_diesel_inputs.fuel_cost_L = -1;
43
44             Diesel bad_diesel(8760, bad_diesel_inputs);
45
46             error_flag = false;
47         } catch (...) {
48             // Task failed successfully! =P
49         }
50         if (not error_flag) {
51             expectedErrorNotDetected(__FILE__, __LINE__);
52         }
53
54
55         DieselInputs diesel_inputs;
56
57         Diesel test_diesel(8760, diesel_inputs);
58
59         // 2. test structure attributes
60         testTruth(
61             not diesel_inputs.combustion_inputs.production_inputs.print_flag,
62             __FILE__,
63             __LINE__
64         );
65
66
67         // 3. test post-construction attributes
68         testFloatEquals(
69             test_diesel.linear_fuel_slope_LkWh,
70             0.265675,
71             __FILE__,
72             __LINE__
73         );
74
75         testFloatEquals(
76             test_diesel.linear_fuel_intercept_LkWh,
77             0.026676,
```

```

78         __FILE__,
79         __LINE__
80     );
81
82     testFloatEquals(
83         test_diesel.capital_cost,
84         67846.467018,
85         __FILE__,
86         __LINE__
87     );
88
89     testFloatEquals(
90         test_diesel.operation_maintenance_cost_kWh,
91         0.038027,
92         __FILE__,
93         __LINE__
94     );
95
96     testFloatEquals(
97         test_diesel.minimum_load_ratio,
98         0.2,
99         __FILE__,
100        __LINE__
101    );
102
103    testFloatEquals(
104        test_diesel.minimum_runtime_hrs,
105        1,
106        __FILE__,
107        __LINE__
108    );
109
110
111    // 4. test methods
112
113    testFloatEquals(
114        test_diesel.requestProductionkW(0, 1, 2 * test_diesel.capacity_kW),
115        test_diesel.capacity_kW,
116        __FILE__,
117        __LINE__
118    );
119
120    testFloatEquals(
121        test_diesel.requestProductionkW(
122            0,
123            1,
124            0.5 * test_diesel.minimum_load_ratio * test_diesel.capacity_kW
125        ),
126        test_diesel.minimum_load_ratio * test_diesel.capacity_kW,
127        __FILE__,
128        __LINE__
129    );
130 }
131
132 catch (...) {
133     //...
134
135     printGold(" ... ");
136     printRed("FAIL");
137     std::cout << std::endl;
138     throw;
139 }
140
141
142 printGold(" ... ");
143 printGreen("PASS");
144 std::cout << std::endl;
145 return 0;
146 } /* main() */

```

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

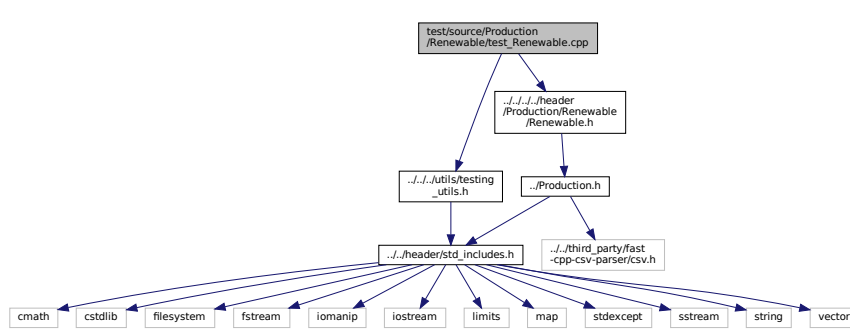
Testing suite for [Renewable](#) class.

```

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

```

Include dependency graph for test_Renewable.cpp:



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 )
{
    27 {
    28     #ifdef _WIN32
    29         activateVirtualTerminal();
    30     #endif /* _WIN32 */
    31
    32     printGold("\tTesting Production <-- Renewable");
    33
    34     srand(time(NULL));
    35
    36
    37     try {
    38         //...
    39     }
    40
    41     catch (...) {
    42         //...
    43
    44         printGold(" ..... ");
    45         printRed("FAIL");
    46         std::cout << std::endl;
    47         throw;
    48     }
    49
    50
    51     printGold(" ..... ");
    52     printGreen("PASS");
    53     std::cout << std::endl;
    54     return 0;
    55 } /* main() */

```

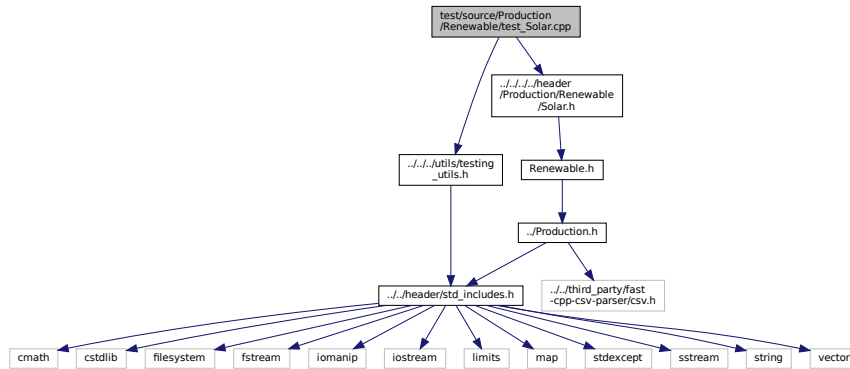
5.34 test/source/Production/Renewable/test_Solar.cpp File Reference

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 )
{
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Production <-- Renewable <-- Solar");
33
34     srand(time(NULL));
35
36
37     try {
38         //...

```

```

39     }
40
41     catch (...) {
42         //...
43
44         printGold(" ..... ");
45         printRed("FAIL");
46         std::cout << std::endl;
47         throw;
48     }
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* main() */

```

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

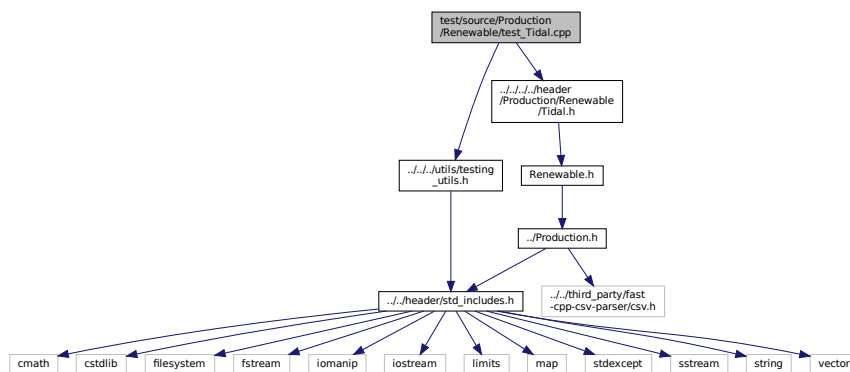
Testing suite for [Tidal](#) class.

```

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

```

Include dependency graph for test_Tidal.cpp:



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 )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Production <-- Renewable <-- Tidal");
33
34     srand(time(NULL));
35
36
37     try {
38         //...
39     }
40
41     catch (...) {
42         //...
43
44         printGold(" ..... ");
45         printRed("FAIL");
46         std::cout << std::endl;
47         throw;
48     }
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* main() */

```

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

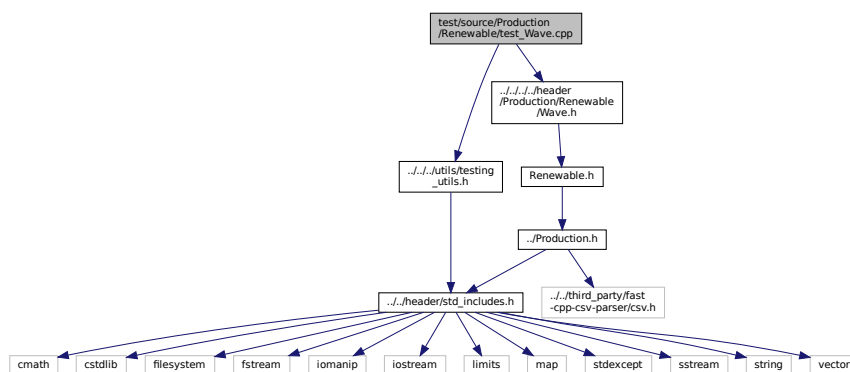
Testing suite for [Wave](#) class.

```

#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()

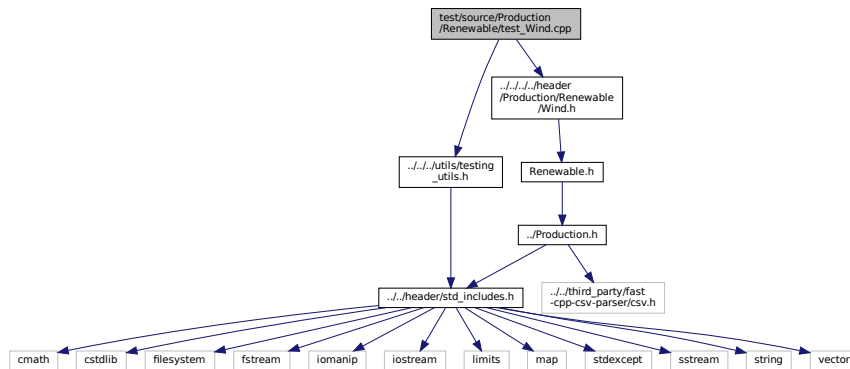
```
int main (
    int argc,
    char ** argv )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Production <-- Renewable <-- Wave");
33
34     srand(time(NULL));
35
36     try {
37         //...
38     }
39
40     catch (...) {
41         //...
42
43         printGold(" ..... ");
44         printRed("FAIL");
45         std::cout << std::endl;
46         throw;
47     }
48
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* 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()

```

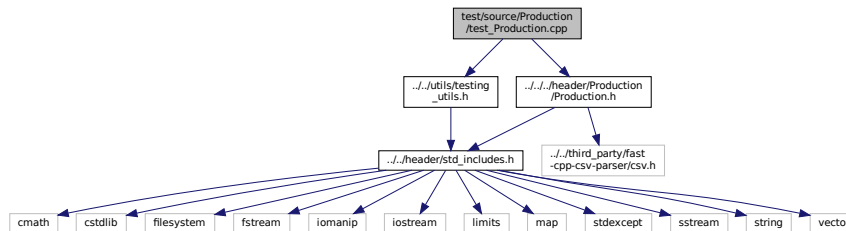
int main (
    int argc,
    char ** argv )
{
    27 {
    28     #ifdef _WIN32
    29         activateVirtualTerminal();
    30     #endif /* _WIN32 */
    31
    32     printGold("\tTesting Production <-- Renewable <-- Wind");
    33
    34     srand(time(NULL));
    35
    36
    37     try {
    38         //...
    39     }
    40
    41     catch (...) {
    42         //...
    43
    44         printGold(" ..... ");
    45         printRed("FAIL");
    46         std::cout << std::endl;
    47         throw;
    48     }
    49
    50
    51     printGold(" ..... ");
    52     printGreen("PASS");
    53     std::cout << std::endl;
    54     return 0;
    55 } /* 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 )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\n\tTesting Production");
33
34     srand(time(NULL));
35
36
37     try {
38         // 1. construction
39         bool error_flag = true;
40         try {
41             ProductionInputs production_inputs;
42
43             Production bad_production(0, production_inputs);
44
```

```

45         error_flag = false;
46     } catch (...) {
47         // Task failed successfully! =P
48     }
49     if (not error_flag) {
50         expectedErrorNotDetected(__FILE__, __LINE__);
51     }
52
53
54     ProductionInputs production_inputs;
55
56     Production test_production(8760, production_inputs);
57
58
59     // 2. test structure attributes
60     testTruth(
61         not production_inputs.print_flag,
62         __FILE__,
63         __LINE__
64     );
65
66     testFloatEquals(
67         production_inputs.nominal_inflation_annual,
68         0.02,
69         __FILE__,
70         __LINE__
71     );
72
73     testFloatEquals(
74         production_inputs.nominal_discount_annual,
75         0.04,
76         __FILE__,
77         __LINE__
78     );
79
80
81     // 3. test post-construction attributes
82     testFloatEquals(
83         test_production.n_points,
84         8760,
85         __FILE__,
86         __LINE__
87     );
88
89     testFloatEquals(
90         test_production.capacity_kW,
91         100,
92         __FILE__,
93         __LINE__
94     );
95
96     testFloatEquals(
97         test_production.real_discount_annual,
98         0.0196078431372549,
99         __FILE__,
100        __LINE__
101    );
102
103    testFloatEquals(
104        test_production.production_vec_kW.size(),
105        8760,
106        __FILE__,
107        __LINE__
108    );
109
110    testFloatEquals(
111        test_production.dispatch_vec_kW.size(),
112        8760,
113        __FILE__,
114        __LINE__
115    );
116
117    testFloatEquals(
118        test_production.storage_vec_kW.size(),
119        8760,
120        __FILE__,
121        __LINE__
122    );
123
124    testFloatEquals(
125        test_production.curtailement_vec_kW.size(),
126        8760,
127        __FILE__,
128        __LINE__
129    );
130
131    testFloatEquals(

```

```

132         test_production.capital_cost_vec.size(),
133         8760,
134         __FILE__,
135         __LINE__
136     );
137
138     testFloatEquals(
139         test_production.operation_maintenance_cost_vec.size(),
140         8760,
141         __FILE__,
142         __LINE__
143     );
144 }
145
146 catch (...) {
147     //...
148
149     printGold(" ..... ");
150     printRed("FAIL");
151     std::cout << std::endl;
152     throw;
153 }
154
155
156 printGold(" ..... ");
157 printGreen("PASS");
158 std::cout << std::endl;
159 return 0;
160 } /* main() */

```

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

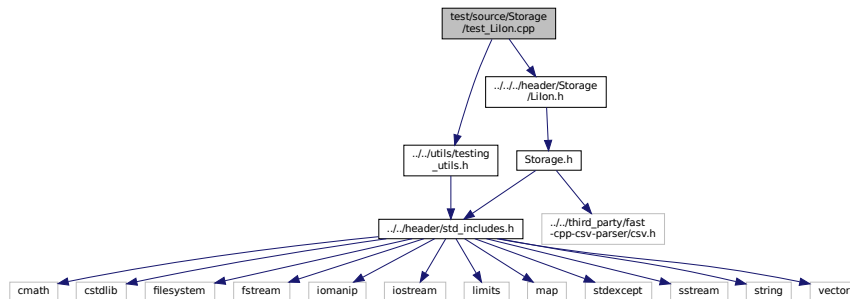
Testing suite for [Lilon](#) class.

```

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

```

Include dependency graph for test_Lilon.cpp:



Functions

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

5.39.1 Detailed Description

Testing suite for [Lilon](#) class.

A suite of tests for the [Lilon](#) class.

5.39.2 Function Documentation

5.39.2.1 main()

```

int main (
    int argc,
    char ** argv )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Storage <-- LiIon");
33
34     srand(time(NULL));
35
36
37     try {
38         //...
39     }
40
41     catch (...) {
42         //...
43
44         printGold(" ..... ");
45         printRed("FAIL");
46         std::cout << std::endl;
47         throw;
48     }
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* main() */

```

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

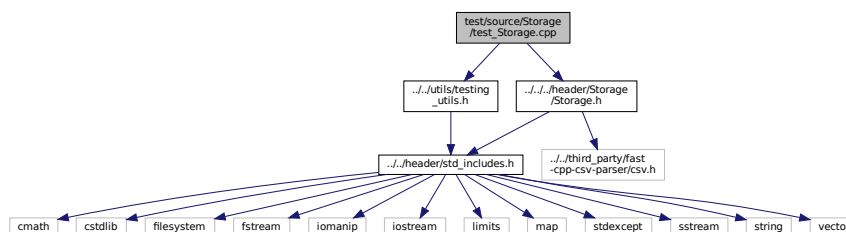
Testing suite for [Storage](#) class.

```

#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

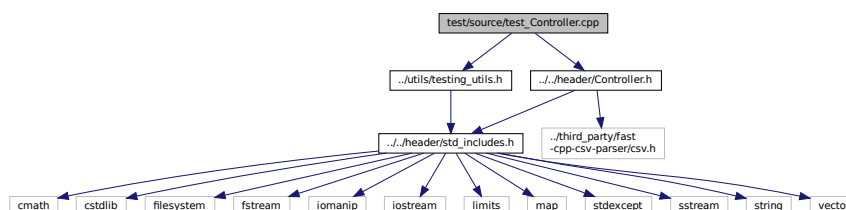
5.40.2.1 main()

```
int main (
    int argc,
    char ** argv )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Storage");
33
34     srand(time(NULL));
35
36
37     try {
38         //...
39     }
40
41     catch (...) {
42         //...
43
44         printGold(" ..... ");
45         printRed("FAIL");
46         std::cout << std::endl;
47         throw;
48     }
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* 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

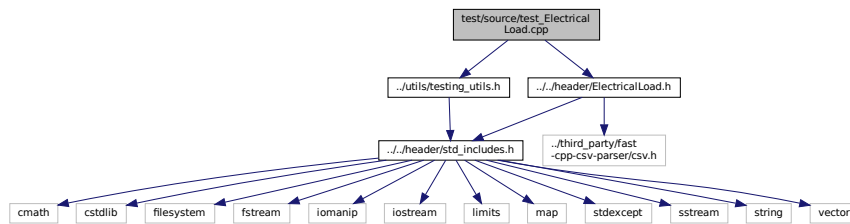
5.41.2.1 main()

```
int main (
    int argc,
    char ** argv )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Controller");
33
34     srand(time(NULL));
35
36
37     try {
38         //...
39     }
40
41     catch (...) {
42         //...
43
44         printGold(" ..... ");
45         printRed("FAIL");
46         std::cout << std::endl;
47         throw;
48     }
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* 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 )
{
    27 {
    28     #ifdef _WIN32
    29         activateVirtualTerminal();
    30     #endif /* _WIN32 */
    31
    32     printGold("\tTesting ElectricalLoad");
    33
    34     srand(time(NULL));
    35
    36
    37     try {
    38         //...
    39     }
    40
    41     catch (...) {
    42         //...
    43
    44         printGold(" ..... ");
    45         printRed("FAIL");
    46         std::cout << std::endl;
    47         throw;
    48     }
    49
    50
    51     printGold(" ..... ");
    52     printGreen("PASS");
    53     std::cout << std::endl;
    54     return 0;
    55 } /* main() */
```

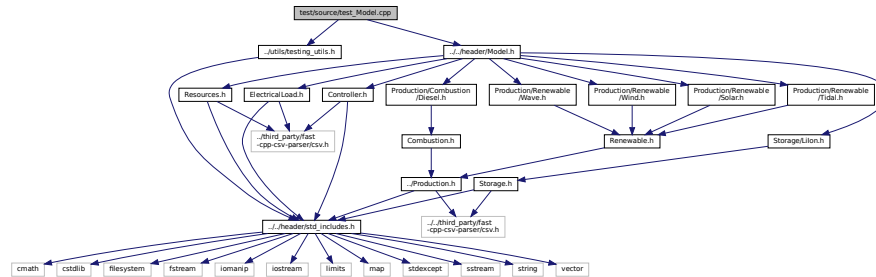
5.43 test/source/test_Model.cpp File Reference

Testing suite for `Model` class.

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

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

Include dependency graph for test_Model.cpp:



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 )

27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Model");
33
34     srand(time(NULL));
35
36
37     try {
38         //...
39     }
40
41     catch (...) {
42         //...
43

```

```

44     printGold(" ..... ");
45     printRed("FAIL");
46     std::cout << std::endl;
47     throw;
48 }
49
50
51 printGold(" ..... ");
52 printGreen("PASS");
53 std::cout << std::endl;
54 return 0;
55 } /* main() */

```

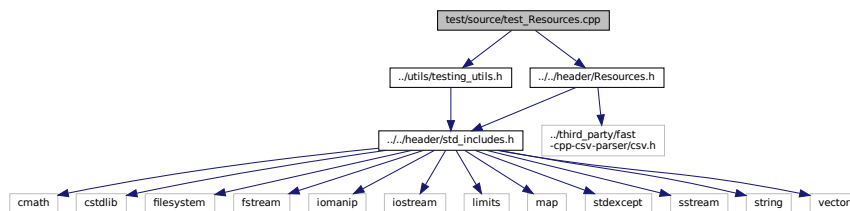
5.44 test/source/test_Resources.cpp File Reference

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 )
27 {
28     #ifdef _WIN32
29         activateVirtualTerminal();
30     #endif /* _WIN32 */
31
32     printGold("\tTesting Resources");
33
34     srand(time(NULL));
35
36
37     try {
38         //...
39     }
40
41     catch (...) {
42         //...
43
44         printGold(" ..... ");
45         printRed("FAIL");
46         std::cout << std::endl;
47         throw;
48     }
49
50
51     printGold(" ..... ");
52     printGreen("PASS");
53     std::cout << std::endl;
54     return 0;
55 } /* main() */

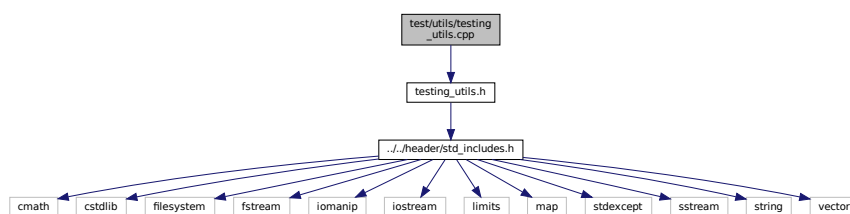
```

5.45 test/utls/testing_utils.cpp File Reference

Header file for various PGMcpp testing utilities.

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

Include dependency graph for testing_utils.cpp:



Functions

- void **printGreen** (std::string input_str)
A function that sends green text to std::cout.
- void **printGold** (std::string input_str)
A function that sends gold text to std::cout.
- void **printRed** (std::string input_str)
A function that sends red text to std::cout.
- void **testFloatEquals** (double x, double y, std::string file, int line)

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

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

Tests if $x > y$.

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

Tests if $x \geq y$.

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

Tests if $x < y$.

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

Tests if $x \leq y$.

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

Tests if the given statement is true.

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

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

5.45.1 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

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

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

5.45.2.2 printGold()

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

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

Parameters

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

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

5.45.2.3 printGreen()

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

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

Parameters

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

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

5.45.2.4 printRed()

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

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

Parameters

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

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

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

```

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

```

5.45.2.6 testGreaterThan()

```

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

```

Tests if $x > y$.

Parameters

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

```

191 {
192     if (x > y) {
193         return;
194     }
195
196     std::string error_str = "ERROR: testGreaterThan():\t in ";
197     error_str += file;

```



```

198     error_str += "\tline ";
199     error_str += std::to_string(line);
200     error_str += ":\t\n";
201     error_str += std::to_string(x);
202     error_str += " is not greater than ";
203     error_str += std::to_string(y);
204     error_str += "\n";
205
206     #ifdef _WIN32
207         std::cout << error_str << std::endl;
208     #endif
209
210     throw std::runtime_error(error_str);
211     return;
212 } /* testGreaterThan() */

```

5.45.2.7 testGreaterThanOrEqualTo()

```

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

```

Tests if $x \geq y$.

Parameters

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

```

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

```

5.45.2.8 testLessThan()

```

void testLessThan (
    double x,
    double y,

```

```
std::string file,
int line )
```

Tests if $x < y$.

Parameters

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

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

5.45.2.9 testLessThanOrEqualTo()

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

Tests if $x \leq y$.

Parameters

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

```
344 {
345     if (x <= y) {
346         return;
347     }
348
349     std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
350     error_str += file;
351     error_str += "\tline ";
352     error_str += std::to_string(line);
353     error_str += ":\t\n";
354     error_str += std::to_string(x);
355     error_str += " is not less than or equal to ";
```

```

356     error_str += std::to_string(y);
357     error_str += "\n";
358
359     #ifdef _WIN32
360         std::cout << error_str << std::endl;
361     #endif
362
363     throw std::runtime_error(error_str);
364     return;
365 } /* testLessThanOrEqualTo() */

```

5.45.2.10 testTruth()

```

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

```

Tests if the given statement is true.

Parameters

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

```

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

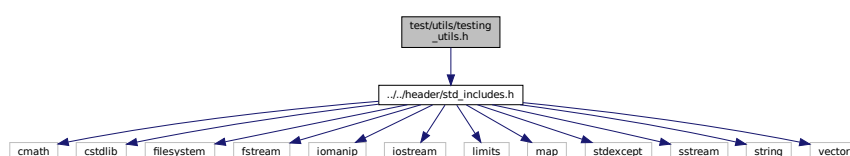
```

5.46 test/utls/testing_utils.h File Reference

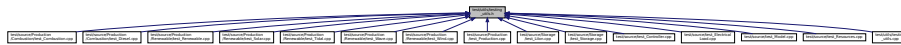
Header file for various PGMcpp testing utilities.

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

Include dependency graph for testing_utils.h:



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



Macros

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

Functions

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

5.46.1 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

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

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

5.46.3.2 printGold()

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

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

Parameters

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

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

5.46.3.3 printGreen()

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

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

Parameters

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

```

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

```

5.46.3.4 printRed()

```

void printRed (
    std::string input_str )

```

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

Parameters

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

```

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

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

```

138 {
139     if (fabs(x - y) <= FLOAT_TOLERANCE) {
140         return;
141     }
142
143     std::string error_str = "ERROR: testFloatEquals():\t in ";
144     error_str += file;
145     error_str += "\tline ";
146     error_str += std::to_string(line);
147     error_str += ":\t\n";
148     error_str += std::to_string(x);
149     error_str += " and ";
150     error_str += std::to_string(y);
151     error_str += " are not equal to within +/- ";

```

```

152     error_str += std::to_string(FLOAT_TOLERANCE);
153     error_str += "\n";
154
155     #ifdef _WIN32
156         std::cout << error_str << std::endl;
157     #endif
158
159     throw std::runtime_error(error_str);
160     return;
161 } /* testFloatEquals() */

```

5.46.3.6 testGreaterThan()

```

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

```

Tests if $x > y$.

Parameters

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

```

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

```

5.46.3.7 testGreaterThanOrEqualTo()

```

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

```

Tests if $x \geq y$.

Parameters

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

```

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

```

5.46.3.8 testLessThan()

```

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

```

Tests if $x < y$.

Parameters

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

```

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

```



```

313     return;
314 } /* testLessThan() */

```

5.46.3.9 testLessThanOrEqualTo()

```

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

```

Tests if $x \leq y$.

Parameters

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

```

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

```

5.46.3.10 testTruth()

```

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

```

Tests if the given statement is true.

Parameters

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

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

Index

- ~Combustion
 - Combustion, [10](#)
- ~Controller
 - Controller, [15](#)
- ~Diesel
 - Diesel, [19](#)
- ~ElectricalLoad
 - ElectricalLoad, [27](#)
- ~Lilon
 - Lilon, [30](#)
- ~Model
 - Model, [32](#)
- ~Production
 - Production, [37](#)
- ~Renewable
 - Renewable, [44](#)
- ~Resources
 - Resources, [45](#)
- ~Solar
 - Solar, [47](#)
- ~Storage
 - Storage, [49](#)
- ~Tidal
 - Tidal, [50](#)
- ~Wave
 - Wave, [52](#)
- ~Wind
 - Wind, [54](#)
- capacity_kW
 - Production, [38](#)
 - ProductionInputs, [42](#)
- capital_cost
 - DieselInputs, [24](#)
 - Production, [38](#)
- capital_cost_vec
 - Production, [38](#)
- CH4_emissions_intensity_kgL
 - Diesel, [21](#)
 - DieselInputs, [24](#)
- CH4_emissions_vec_kg
 - Combustion, [12](#)
- CH4_kg
 - Emissions, [28](#)
- CO2_emissions_intensity_kgL
 - Diesel, [21](#)
 - DieselInputs, [24](#)
- CO2_emissions_vec_kg
 - Combustion, [12](#)
- CO2_kg
 - Emissions, [28](#)
- CO_emissions_intensity_kgL
 - Diesel, [21](#)
 - DieselInputs, [24](#)
- CO_emissions_vec_kg
 - Combustion, [12](#)
- CO_kg
 - Emissions, [28](#)
- Combustion, [7](#)
 - ~Combustion, [10](#)
 - CH4_emissions_vec_kg, [12](#)
 - CO2_emissions_vec_kg, [12](#)
 - CO_emissions_vec_kg, [12](#)
 - Combustion, [9](#)
 - commit, [10](#)
 - fuel_consumption_vec_L, [12](#)
 - fuel_cost_vec, [13](#)
 - getEmissionskg, [11](#)
 - getFuelConsumptionL, [11](#)
 - linear_fuel_intercept_LkWh, [13](#)
 - linear_fuel_slope_LkWh, [13](#)
 - NOx_emissions_vec_kg, [13](#)
 - PM_emissions_vec_kg, [13](#)
 - requestProductionkW, [12](#)
 - SOx_emissions_vec_kg, [13](#)
- Combustion.h
 - CombustionType, [61](#)
 - DIESEL, [61](#)
 - N_COMBUSTION_TYPES, [61](#)
- combustion_inputs
 - DieselInputs, [25](#)
- combustion_ptr_vec
 - Model, [33](#)
- CombustionInputs, [14](#)
 - production_inputs, [14](#)
- CombustionType
 - Combustion.h, [61](#)
- commit
 - Combustion, [10](#)
 - Diesel, [19](#)
 - Production, [37](#)
- Controller, [15](#)
 - ~Controller, [15](#)
 - Controller, [15](#)
- controller
 - Model, [33](#)
- curtailment_vec_kW
 - Production, [38](#)
- DIESEL

- Combustion.h, 61
- Diesel, 16
 - ~Diesel, 19
 - CH4_emissions_intensity_kgL, 21
 - CO2_emissions_intensity_kgL, 21
 - CO_emissions_intensity_kgL, 21
 - commit, 19
 - Diesel, 18
 - fuel_cost_L, 21
 - minimum_load_ratio, 21
 - minimum_runtime_hrs, 21
 - NOx_emissions_intensity_kgL, 22
 - PM_emissions_intensity_kgL, 22
 - requestProductionkW, 20
 - SOx_emissions_intensity_kgL, 22
 - time_since_last_start_hrs, 22
- DieselInputs, 23
 - capital_cost, 24
 - CH4_emissions_intensity_kgL, 24
 - CO2_emissions_intensity_kgL, 24
 - CO_emissions_intensity_kgL, 24
 - combustion_inputs, 25
 - fuel_cost_L, 25
 - linear_fuel_intercept_LkWh, 25
 - linear_fuel_slope_LkWh, 25
 - minimum_load_ratio, 25
 - minimum_runtime_hrs, 25
 - NOx_emissions_intensity_kgL, 26
 - operation_maintenance_cost_kWh, 26
 - PM_emissions_intensity_kgL, 26
 - SOx_emissions_intensity_kgL, 26
- dispatch_vec_kW
 - Production, 39
- electrical_load
 - Model, 33
- ElectricalLoad, 26
 - ~ElectricalLoad, 27
 - ElectricalLoad, 27
- Emissions, 27
 - CH4_kg, 28
 - CO2_kg, 28
 - CO_kg, 28
 - NOx_kg, 28
 - PM_kg, 29
 - SOx_kg, 29
- expectedErrorNotDetected
 - testing_utils.cpp, 100
 - testing_utils.h, 107
- FLOAT_TOLERANCE
 - testing_utils.h, 106
- fuel_consumption_vec_L
 - Combustion, 12
- fuel_cost_L
 - Diesel, 21
 - DieselInputs, 25
- fuel_cost_vec
 - Combustion, 13
- getEmissionskg
 - Combustion, 11
- getFuelConsumptionL
 - Combustion, 11
- header/Controller.h, 57
- header/ElectricalLoad.h, 58
- header/Model.h, 59
- header/Production/Combustion/Combustion.h, 60
- header/Production/Combustion/Diesel.h, 61
- header/Production/Production.h, 62
- header/Production/Renewable/Renewable.h, 63
- header/Production/Renewable/Solar.h, 63
- header/Production/Renewable/Tidal.h, 64
- header/Production/Renewable/Wave.h, 65
- header/Production/Renewable/Wind.h, 66
- header/Resources.h, 67
- header/std_includes.h, 68
- header/Storage/Lilon.h, 69
- header/Storage/Storage.h, 69
- is_running
 - Production, 39
- is_running_vec
 - Production, 39
- is_sunk
 - Production, 39
 - ProductionInputs, 42
- levellized_cost_of_energy_kWh
 - Production, 39
- Lilon, 29
 - ~Lilon, 30
 - Lilon, 30
- linear_fuel_intercept_LkWh
 - Combustion, 13
 - DieselInputs, 25
- linear_fuel_slope_LkWh
 - Combustion, 13
 - DieselInputs, 25
- main
 - test_Combustion.cpp, 80
 - test_Controller.cpp, 95
 - test_Diesel.cpp, 82
 - test_ElectricalLoad.cpp, 96
 - test_Lilon.cpp, 93
 - test_Model.cpp, 97
 - test_Production.cpp, 90
 - test_Renewable.cpp, 84
 - test_Resources.cpp, 98
 - test_Solar.cpp, 85
 - test_Storage.cpp, 94
 - test_Tidal.cpp, 86
 - test_Wave.cpp, 88
 - test_Wind.cpp, 89
- minimum_load_ratio
 - Diesel, 21
 - DieselInputs, 25

- minimum_runtime_hrs
 - Diesel, [21](#)
 - DieselInputs, [25](#)
- Model, [31](#)
 - ~Model, [32](#)
 - combustion_ptr_vec, [33](#)
 - controller, [33](#)
 - electrical_load, [33](#)
 - Model, [32](#)
 - renewable_ptr_vec, [33](#)
 - resources, [33](#)
 - storage_ptr_vec, [33](#)
- N_COMBUSTION_TYPES
 - Combustion.h, [61](#)
- n_points
 - Production, [39](#)
- n_starts
 - Production, [40](#)
- net_present_cost
 - Production, [40](#)
- nominal_discount_annual
 - ProductionInputs, [42](#)
- nominal_inflation_annual
 - ProductionInputs, [42](#)
- NOx_emissions_intensity_kgL
 - Diesel, [22](#)
 - DieselInputs, [26](#)
- NOx_emissions_vec_kg
 - Combustion, [13](#)
- NOx_kg
 - Emissions, [28](#)
- operation_maintenance_cost_kWh
 - DieselInputs, [26](#)
 - Production, [40](#)
- operation_maintenance_cost_vec
 - Production, [40](#)
- PM_emissions_intensity_kgL
 - Diesel, [22](#)
 - DieselInputs, [26](#)
- PM_emissions_vec_kg
 - Combustion, [13](#)
- PM_kg
 - Emissions, [29](#)
- print_flag
 - Production, [40](#)
 - ProductionInputs, [43](#)
- printGold
 - testing_utils.cpp, [100](#)
 - testing_utils.h, [107](#)
- printGreen
 - testing_utils.cpp, [101](#)
 - testing_utils.h, [107](#)
- printRed
 - testing_utils.cpp, [101](#)
 - testing_utils.h, [108](#)
- Production, [34](#)
 - ~Production, [37](#)
 - capacity_kW, [38](#)
 - capital_cost, [38](#)
 - capital_cost_vec, [38](#)
 - commit, [37](#)
 - curtailment_vec_kW, [38](#)
 - dispatch_vec_kW, [39](#)
 - is_running, [39](#)
 - is_running_vec, [39](#)
 - is_sunk, [39](#)
 - levellized_cost_of_energy_kWh, [39](#)
 - n_points, [39](#)
 - n_starts, [40](#)
 - net_present_cost, [40](#)
 - operation_maintenance_cost_kWh, [40](#)
 - operation_maintenance_cost_vec, [40](#)
 - print_flag, [40](#)
 - Production, [36](#)
 - production_vec_kW, [40](#)
 - real_discount_annual, [41](#)
 - running_hours, [41](#)
 - storage_vec_kW, [41](#)
- production_inputs
 - CombustionInputs, [14](#)
- production_vec_kW
 - Production, [40](#)
- ProductionInputs, [41](#)
 - capacity_kW, [42](#)
 - is_sunk, [42](#)
 - nominal_discount_annual, [42](#)
 - nominal_inflation_annual, [42](#)
 - print_flag, [43](#)
- PYBIND11_MODULE
 - PYBIND11_PGM.cpp, [71](#)
- PYBIND11_PGM.cpp
 - PYBIND11_MODULE, [71](#)
- pybindings/PYBIND11_PGM.cpp, [70](#)
- real_discount_annual
 - Production, [41](#)
- Renewable, [43](#)
 - ~Renewable, [44](#)
 - Renewable, [44](#)
- renewable_ptr_vec
 - Model, [33](#)
- requestProductionkW
 - Combustion, [12](#)
 - Diesel, [20](#)
- Resources, [45](#)
 - ~Resources, [45](#)
 - Resources, [45](#)
- resources
 - Model, [33](#)
- running_hours
 - Production, [41](#)
- Solar, [46](#)
 - ~Solar, [47](#)
 - Solar, [47](#)

- source/Controller.cpp, 72
- source/ElectricalLoad.cpp, 73
- source/Model.cpp, 73
- source/Production/Combustion/Combustion.cpp, 74
- source/Production/Combustion/Diesel.cpp, 74
- source/Production/Production.cpp, 75
- source/Production/Renewable/Renewable.cpp, 75
- source/Production/Renewable/Solar.cpp, 76
- source/Production/Renewable/Tidal.cpp, 76
- source/Production/Renewable/Wave.cpp, 77
- source/Production/Renewable/Wind.cpp, 77
- source/Resources.cpp, 78
- source/Storage/Lilon.cpp, 78
- source/Storage/Storage.cpp, 79
- SOx_emissions_intensity_kgL
 - Diesel, 22
 - DieselInputs, 26
- SOx_emissions_vec_kg
 - Combustion, 13
- SOx_kg
 - Emissions, 29
- Storage, 48
 - ~Storage, 49
 - Storage, 49
- storage_ptr_vec
 - Model, 33
- storage_vec_kW
 - Production, 41
- test/source/Production/Combustion/test_Combustion.cpp, 79
- test/source/Production/Combustion/test_Diesel.cpp, 81
- test/source/Production/Renewable/test_Renewable.cpp, 83
- test/source/Production/Renewable/test_Solar.cpp, 85
- test/source/Production/Renewable/test_Tidal.cpp, 86
- test/source/Production/Renewable/test_Wave.cpp, 87
- test/source/Production/Renewable/test_Wind.cpp, 88
- test/source/Production/test_Production.cpp, 90
- test/source/Storage/test_Lilon.cpp, 92
- test/source/Storage/test_Storage.cpp, 93
- test/source/test_Controller.cpp, 94
- test/source/test_ElectricalLoad.cpp, 95
- test/source/test_Model.cpp, 97
- test/source/test_Resources.cpp, 98
- test/utls/testing_utils.cpp, 99
- test/utls/testing_utils.h, 105
- test_Combustion.cpp
 - main, 80
- test_Controller.cpp
 - main, 95
- test_Diesel.cpp
 - main, 82
- test_ElectricalLoad.cpp
 - main, 96
- test_Lilon.cpp
 - main, 93
- test_Model.cpp
 - main, 97
- test_Production.cpp
 - main, 90
- test_Renewable.cpp
 - main, 84
- test_Resources.cpp
 - main, 98
- test_Solar.cpp
 - main, 85
- test_Storage.cpp
 - main, 94
- test_Tidal.cpp
 - main, 86
- test_Wave.cpp
 - main, 88
- test_Wind.cpp
 - main, 89
- testFloatEquals
 - testing_utils.cpp, 101
 - testing_utils.h, 108
- testGreaterThan
 - testing_utils.cpp, 102
 - testing_utils.h, 109
- testGreaterThanOrEqualTo
 - testing_utils.cpp, 103
 - testing_utils.h, 109
- testing_utils.cpp
 - expectedErrorNotDetected, 100
 - printGold, 100
 - printGreen, 101
 - printRed, 101
 - testFloatEquals, 101
 - testGreaterThan, 102
 - testGreaterThanOrEqualTo, 103
 - testLessThan, 103
 - testLessThanOrEqualTo, 104
 - testTruth, 105
- testing_utils.h
 - expectedErrorNotDetected, 107
 - FLOAT_TOLERANCE, 106
 - printGold, 107
 - printGreen, 107
 - printRed, 108
 - testFloatEquals, 108
 - testGreaterThan, 109
 - testGreaterThanOrEqualTo, 109
 - testLessThan, 110
 - testLessThanOrEqualTo, 111
 - testTruth, 111
- testLessThan
 - testing_utils.cpp, 103
 - testing_utils.h, 110
- testLessThanOrEqualTo
 - testing_utils.cpp, 104
 - testing_utils.h, 111
- testTruth
 - testing_utils.cpp, 105
 - testing_utils.h, 111
- Tidal, 49

~Tidal, [50](#)
Tidal, [50](#)
time_since_last_start_hrs
Diesel, [22](#)

Wave, [51](#)
~Wave, [52](#)
Wave, [52](#)
Wind, [53](#)
~Wind, [54](#)
Wind, [54](#)