# Simopticon

1.0

Generated by Doxygen 1.9.1

Fri Aug 4 2023 23:13:27

# Chapter 1

# Simopticon



## 1.1 Overview

*Simopticon* is a framework which automates the search for optimal parameters for simulated processes. The key strategy is to define parameters that shall be optimized, automatically run simulations with certain parameters, evaluate their performance by calculating a number rating (the lower, the better) and trying to find parameter combinations that minimize the rating.
The described process is distributed over four major components:

1. Optimizer: An optimization strategy capable of finding the minimum of a blackbox function only accessible through argument-value pairs.
2. SimulationRunner: A component used to run simulations with certain parameters automatically.
3. Evaluation: A component capable of calculating a rating value based on result files of simulations.
4. Controller: A component managing the optimization process and communication between Optimizer, SimulationRunner and Evaluation. Used to abstract components 1-3 from each other.

Extensions of the framework may introduce new Optimizer, SimulationRunner and Evaluation implementations (see Extension). Currently, there is only one implementation of each component, tailored for the optimization of platoon controllers using the Plexe framework.
The full API documentation may be found on peternaggschga.github.io/simopticon.

## 1.2 Setup

### 1.2.1 Requirements

The following sections describe the requirements your machine has to fulfill to run *Simopticon*. They may differ depending on the Optimizer, SimulationRunner and Evaluation implementations you plan to use, therefore, the implementations have their own dependency sections.

#### Simopticon

The framework itself is developed for Debian-based Unix/Linux machines. Other operating systems might work but are not actively supported. To be able to install the framework, you need the following software:

- Git (see `Git`)
- CMake Version 3.25 or higher (see `CMake`)
- Python3 development tools (see `Python3 Development Tools`)

#### PlexeSimulationRunner

To enable simulations with Plexe, Version 3.1 of the framework must be installed. Refer to the `Plexe install guide` for more information. Please mind that you might want to install OMNeT++ Version 6 or higher in order to use the `ConstantHeadway` Evaluation, even though the installation guide might suggest an older version.

#### ConstantHeadway

To use the ConstantHeadway Evaluation, OMNeT++ Version 6 or higher is needed. Please refer to the `OMNeT++ Install Guide` for more information on the requirements.

### 1.2.2 Installation

#### Prerequisites

**Git** Check whether Git is installed on your machine and install it if necessary using:
```
sudo apt install git
```

**CMake** CMake Version 3.25 or higher is needed for building *Simopticon*. If you don't have CMake installed, follow the guide below. If you have an older version installed, you must first remove it.
First, make sure to install g++ and OpenSSL Development tools.
```
sudo apt install g++ libssl-dev
```
Then you need to download the latest version of CMake from their `download page` — search for the source distribution tar package. Unpack the downloaded package using:
```
tar xf cmake-[version number].tar.gz
```
Open the newly created directory and run the configuration script with:
```
cd cmake-[version number] && ./configure
```
When the configuration has completed successfully, you are ready to build and install using:
```
make -j $(nproc)
sudo make install
```
You may remove the downloaded tar file and extracted directory if needed.

**Python3 Development Tools** Check whether Python3 development tools are installed on your machine and install them if necessary using:
```
sudo apt install python3-dev
```

#### Simopticon

Go to the directory you want to install *Simopticon* in, e.g. `~/src`. To get the source code, clone the git repository using:
```
git clone https://github.com/PeterNaggschga/simopticon.git
```
Create a build directory in the downloaded files with:
```
mkdir simopticon/build
cd simopticon/build
```

Build *Simopticon* by calling:
```
cmake ..
make -j $(nproc)
```
The resulting executable `simopticon` may be copied to other locations or referenced via symlinks for more convenient access. The same applies to the `config` directory in `~/src/simopticon` which is used to configure the optimization process (see Usage).

## 1.3 Usage

### 1.3.1 Configuration

The optimization process and its components are configured using several JSON files. Default examples of such files can be found in the `config` directory. Be aware, however, that the default files in `config` must be edited before use, since some file paths must be set which depend on your filesystem.
The options in the JSON files are commented and therefore self-explanatory. The following sections only show options that must be changed to successfully run optimizations.

#### Main Configuration

The main configuration can be found in `config/simopticon.json`. It contains settings of the Controller and selects the other components. In the `controller` settings, the key `params` must be set to reference another JSON file containing an array of ParameterDefinition that are to be optimized.
The main configuration selects which Optimizer, SimulationRunner and Evaluation implementations are to be used. For each of those components, a name of the implementation and a reference to a JSON file configuring it must be given. References are used because different implementations of the same component may vastly differ in their configurable options, and switching the used components gets easier this way.

#### PlexeSimulationRunner

If you want to use PlexeSimulationRunner, you need to configure `config/runners/plexe.json`. There you have to set the `configDirectory` key to match the path to the directory containing your Plexe configuration (`omnetpp.ini`). For default installations that should be something along the lines of `[installation-directory]/plexe/examples/platooning`.

#### ConstantHeadway

If you want to use ConstantHeadway evaluation, you need to configure `config/evaluations/constant_↩headway.json`. There you have to set the `pythonScript` and the `omnetppDirectory` keys. `python↩Script` must point to the script `constant_headway.py` which can be found in `src/evaluation/constant↩_headway`. `omnetppDirectory` must point to the directory where OMNeT++ Version 6 or higher is installed, e.g. `~/src/omnetpp-6.0.1`.

### 1.3.2 Optimization

The optimization is invoked on the command line by executing the program built in Setup. The call on the command line has one mandatory and one optional argument. The First argument must be the path to the main config, i.e. `config/simopticon.json`. A valid call to an optimization could be:
```
./simopticon ../config/simopticon.json
```
If a second argument is given, instead of running actual simulations with the configured SimulationRunner and evaluating their results with an Evaluation, the StubController is used. StubController can be used to implement and optimize benchmark functions to test Optimizer implementations without relying on actual costly simulations. The second argument holds the name of the function to be optimized, i.e., one of the following:

- quadratic (squares all Parameter values and adds them up)
- `branin`
- `goldprice`
- `camel6`
- `shubert`

- hartman3
- shekel5
- shekel7
- shekel10
- hartman6

A valid call to the optimization of a benchmark function could be:
```
./simopticon ../config/simopticon.json branin
```
Please note that you need to define the optimized parameters in `config/simopticon.json` even when you are optimizing a benchmark.

## 1.4 Extension

This section goes through the steps you need to undertake to extend the framework with new Optimizer, SimulationRunner or Evaluation implementations.

### 1.4.1 Development

When developing new implementations of components, please stick to the project structure — Optimizer extensions go into `src/optimizer`, SimulationRunner extensions go into `src/runner` and Evaluation extensions go into `src/evaluation`. If your implementation needs a more sophisticated implementation of the Parameter class than the ones provided in `src/parameters`, feel free to extend the abstract Parameter class.
The `src/Types.h` header file defines framework-wide types such as `functionValue` for values returned by the Evaluation component or `coordinate` which is used to store Parameter values. The `src/ComparisonFunctions.h` header file defines comparison functions, which can be used in STL containers that are ordered. E.g. `CmpVectorSharedParameter` can be used to compare two objects of type `vector<shared_ptr<Parameter>>`.

#### Optimization Strategies

To add a new optimization strategy, you have to extend the Optimizer class. You need to override the Optimizer::runOptimization method which should start the optimization process and only return when your strategy is finished or if the Optimizer::abort method is called which you should implement too.
Optimizer extensions can instruct the Controller to start simulations and evaluate them with the Optimizer::requestValues method. Please try to commission as many Parameters as possible in one call of the method so the other components may parallelize calculations.
Please consider overriding the methods provided by the Status interface to give the user a sense of what is happening.

#### Simulation Execution

To add a new way of executing simulations, you have to extend the SimulationRunner class. You need to override the SimulationRunner::work function, which is run concurrently for all Parameter vectors provided to SimulationRunner::runSimulations. If you want to prohibit concurrent execution, you may override SimulationRunner::runSimulations instead (in that case, SimulationRunner::work should return an empty pair). See documentation of Multithreaded class for more information on that.
SimulationRunner::work should run a simulation with the given parameters and return a path to the result files and a set of identifiers relating to simulation runs. The interface for the identifiers is very loosely defined — if your Evaluation does not need any identifiers of simulation runs, you may return an empty set. Please be aware that the Controller might try to delete the path you return after some time, so that should not be an empty path! Other than that, it is not further standardized what must be returned as a path and identifiers as long as your Evaluation component can evaluate the simulation based on the returned information.
Please consider overriding the methods provided by the Status interface to give the user a sense of what is happening.

#### Simulation Evaluation

To add a new rating algorithm based on simulation data, you have to extend the Evaluation class. You need to override the Evaluation::processOutput function, which conducts the rating of simulation performance based on the path to the result files and the given identifiers. This process heavily depends on the implemented SimulationRunner, which is responsible for returning result files and run identifiers if necessary. Your Evaluation implementation should rate the given simulation results with a `functionValue` — the lower, the better.
Please consider overriding the methods provided by the Status interface to give the user a sense of what is happening.

### 1.4.2 Integration

All newly added classes must be registered in `CMakeList.txt` so the compiler does not ignore them! External dependencies and added libraries should be included there too.
To make your new component available for configuration, you must add it to the constructor of the Controller class. Let's assume you wrote a new Optimizer implementation. First you need to create a JSON configuration file in `config/optimizer`. There you can define any desired options for your component.
The next step is editing the Controller class to make your Optimizer available. To do that, you find the "Optimizer settings" in the constructor of the Controller. There you add another case to the `if`-Statement where `opt` equals the name of your component (this is the name that will be set in the main config later, see `Configuration`). In the added case you can read the necessary options from the JSON object in `optimizerConfig`. You have to set Controller::optimizer to an `unique_ptr<Optimizer>`, owning a new instance of your Optimizer implementation.
When this setup is complete, you may build the framework again and update the main configuration to use your new Optimizer by changing the `optimizer.optimizer` key to the name of your Optimizer and the `optimizer.config` key to the path of your created JSON configuration file.

# Chapter 2

# Todo List

**Member interruptHandler ([[maybe_unused]] int s)**

Make interrupt handling independent from OS - currently only Systems using POSIX signals are supported.

# Chapter 3

# Bug List

**Member constant_headway.get_constant_headway (list run_ids)**

Running mean calculation over vectors using omnetpp.scave does not work correctly!

# Chapter 4

# Module Index

## 4.1 Modules

Here is a list of all modules:

# Chapter 5

# Hierarchical Index

## 5.1 Class Hierarchy

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

# Chapter 6

# Class Index

## 6.1 Class List

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

# Chapter 7

# File Index

## 7.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 8

# Module Documentation

## 8.1 Controller

This module provides classes coordinating the optimization process independently from the actual implementation of Optimizer, SimulationRunner and Evaluation.

### Classes

- class Controller

  *A class responsible for communication between Optimizer, SimulationRunner and Evaluation and also user interaction such as tracking results, updating StatusBar and handling interrupts by the user via Abortable.*
- struct Controller::stepstate

  *A struct keeping track of the currently running optimization step for StatusBar::updateStatus.*
- class ValueMap

  *A container managing a map data structure that maps Parameter combinations to their respective found values.*
- class StubController

  *A class that mocks behaviour of Controller.*

### Variables

- struct Controller::stepstate Controller::stepState

  *An object keeping track of the current optimization step.*

### 8.1.1 Detailed Description

This module provides classes coordinating the optimization process independently from the actual implementation of Optimizer, SimulationRunner and Evaluation.

### 8.1.2 Variable Documentation

#### 8.1.2.1 stepState

```
struct Controller::stepstate Controller::stepState  [private]
```

An object keeping track of the current optimization step.

## 8.2 Direct

This module extends Optimizer to use a variant of the DIRECT algorithm by `Jones et al`.

Collaboration diagram for Direct:


## Modules

- Hyrect

  *This module contains the definition of a tree-like data structure representing the partition of a search space into multiple hyper-rectangles (HyRect).*

## Files

- file DirectTypes.h

  *In this file types are defined which are in the direct module.*
- file DirectComparisonFunctions.h

  *In this file comparison functions are defined which are used in the direct module.*

## Classes

- class DirectOptimizer

  *A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.*
- class StoppingCondition

  *A class used for deciding whether the DIRECT should be stopped.*
- class ParameterNormalizer

  *A class used for transforming parameters between the actual Parameter space and the unit hypercube used in DIRECT algorithm.*
- class Levels

  *A providing functionality for the usage of different weightings between local and global search throughout the optimization using different levels.*
- class GrahamScan

  *A class providing functionality for finding the lower right convex hull of a set of points.*

## Enumerations

- enum level : unsigned char {
  **l2_0** = 0 , **l1_1** = 1 , **l0_2** = 2 , **l1_3** = 3 ,
  **l1_4** = 4 , **l0_5** = 5 , **l1_6** = 6 , **l2_7** = 7 }

  *An enum representing the sequence of local levels.*

### 8.2.1 Detailed Description

This module extends Optimizer to use a variant of the DIRECT algorithm by `Jones et al.`.
It incorporates features proposed by `Liu et al.` and `Sergeyev and Kvasov`.

### 8.2.2 Enumeration Type Documentation

#### 8.2.2.1 level

`enum level : unsigned char`
An enum representing the sequence of local levels.
Definition at line 18 of file Levels.h.

## 8.3 Plexe

This module extends SimulationRunner to interface with the Plexe framework to enable the optimization of platooning controllers.
Collaboration diagram for Plexe:


### Classes

- class PlexeSimulationRunner

  *A class capable of starting platooning simulations in the `Plexe` framework with given Parameter combinations.*
- class ConfigEditor

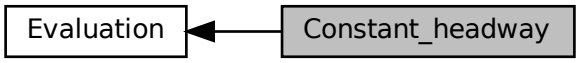
  *A class capable of creating `.ini` files with certain options based on a complete `omnetpp.ini`.*

### 8.3.1 Detailed Description

This module extends SimulationRunner to interface with the Plexe framework to enable the optimization of platooning controllers.

## 8.4 Constant_headway

This module extends Evaluation to interface with a Python script evaluating the performance of platooning simulations with Plexe by analyzing the deviation of vehicles from the pre-specified gap.
Collaboration diagram for Constant_headway:




### Files

- file constant_headway.py

  *A Python script providing functionality for automatic rating of Plexe result files on the mean deviation from the predefined gap.*

### Classes

- class ConstantHeadway

  *A wrapper for the constant_headway.py script.*

### 8.4.1 Detailed Description

This module extends Evaluation to interface with a Python script evaluating the performance of platooning simulations with Plexe by analyzing the deviation of vehicles from the pre-specified gap.

## 8.5 Parameters

This module defines framework-wide representations of the optimized parameters.

### Classes

- class Parameter

  *A class acting as the container of the value of a parameter defined by a ParameterDefinition.*
- class ParameterDefinition

  *A class storing information on the properties of parameters that are being optimized.*
- class ContinuousParameter

  *Implements a Parameter using continuos values in the form of floating point numbers.*
- class DiscreteParameter

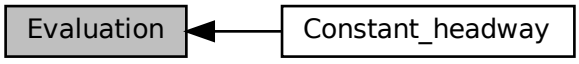
  *Implements a Parameter using discrete values.*

### 8.5.1 Detailed Description

This module defines framework-wide representations of the optimized parameters.

## 8.6 Evaluation

This module contains components capable of evaluating the performance of simulations by rating simulation data with a number value.
Collaboration diagram for Evaluation:




### Modules

- Constant_headway

  *This module extends Evaluation to interface with a Python script evaluating the performance of platooning simulations with Plexe by analyzing the deviation of vehicles from the pre-specified gap.*

### Classes

- class Evaluation

  *A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization.*

### 8.6.1 Detailed Description

This module contains components capable of evaluating the performance of simulations by rating simulation data with a number value.
Implementations must extend Evaluation.

## 8.7 Status

This module provides functionality for command line output to keep the user updated about the optimization state and progress.

### Classes

- class Status

  *An interface defining functions for status updates on configuration and progress of a class.*
- class StatusBar

  *A class used to conduct command line output containing information about the state of the used Optimizer, SimulationRunner and Evaluation along with the found optima.*

### Enumerations

- enum step : char { **INIT** = -1 , **OPTIMIZER** = 0 , **RUNNER** = 1 , **EVALUATION** = 2 }

  *An Enum defining the steps, an optimization process cycles through.*

### 8.7.1 Detailed Description

This module provides functionality for command line output to keep the user updated about the optimization state and progress.

### 8.7.2 Enumeration Type Documentation

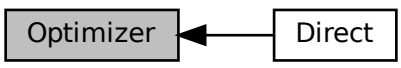
#### 8.7.2.1 step

```
enum step :  char
```

An Enum defining the steps, an optimization process cycles through.
Definition at line 27 of file StatusBar.h.

## 8.8 Optimizer

This module contains components capable of finding the minimum of a function only defined through argument-value pairs.
Collaboration diagram for Optimizer:



### Modules

- Direct

  *This module extends Optimizer to use a variant of the DIRECT algorithm by* `Jones et al`.

### Classes

- class Optimizer

  *A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.*

### 8.8.1 Detailed Description

This module contains components capable of finding the minimum of a function only defined through argument-value pairs.
Implementations must extend Optimizer.

## 8.9 Runner

This module contains components capable of automatically running simulations with certain Parameter combinations.
Collaboration diagram for Runner:


### Modules

- Plexe

  *This module extends SimulationRunner to interface with the Plexe framework to enable the optimization of platooning controllers.*

### Classes

- class SimulationRunner

  *A class capable of running simulations with certain Parameter combinations.*

### 8.9.1 Detailed Description

This module contains components capable of automatically running simulations with certain Parameter combinations.
Implementations must extend SimulationRunner.

## 8.10 Hyrect

This module contains the definition of a tree-like data structure representing the partition of a search space into multiple hyper-rectangles (HyRect).

Collaboration diagram for Hyrect:


## Classes

- class HyRect

  *An abstract class representing a rectangular part of the search space.*
- class BaseRect

  *A class representing a HyRect without a parent rectangle.*
- class ChildRect

  *A class representing a HyRect that has a parent HyRect.*

## Enumerations

- enum class position : char { **LEFT** = 0 , **MIDDLE** = 1 , **RIGHT** = 2 , **BASE** = -1 }

  *An enum representing the position of a HyRect relative to its parent HyRect.*

### 8.10.1 Detailed Description

This module contains the definition of a tree-like data structure representing the partition of a search space into multiple hyper-rectangles (HyRect).

### 8.10.2 Enumeration Type Documentation

#### 8.10.2.1 position

```
enum position : char [strong]
```

An enum representing the position of a HyRect relative to its parent HyRect.
If it is a BaseRect and therefore has no parent, BASE is used.
Definition at line 27 of file HyRect.h.

## 8.11 Utils

This module provides general functionality and classes that may be useful to classes in any other package.

## Files

- file Types.h

  *In this file types are defined which should be used across the whole framework.*
- file ComparisonFunctions.h

  *In this file comparison functions are defined which should be used across the whole framework.*
- file main.cpp

  *Definition of the main function running the Simopticon framework.*

## Classes

- class PythonScript

  *A class containing functionality for interfacing with the function of a Python module on creation.*
- class Abortable

  *A simple interface for classes that encapsulate abortable processes.*
- class Multithreaded< Key, T, Compare, Allocator >

  *A class implementing concurrent execution of the same function for different arguments.*
- class ThreadsafeQueue< Key >

  *A container class of a queue that is safe for concurrent access of different threads.*
- class CommandLine

  *A class containing functionality for executing commands on UNIX shell.*

### 8.11.1 Detailed Description

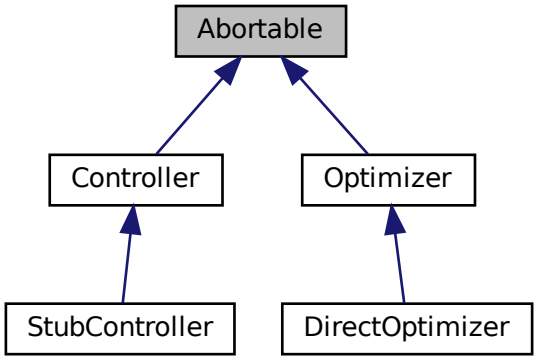
This module provides general functionality and classes that may be useful to classes in any other package.

# Chapter 9

# Class Documentation

## 9.1 Abortable Class Reference

A simple interface for classes that encapsulate abortable processes.
`#include "Abortable.h"`
Inheritance diagram for Abortable:




### Public Member Functions

- virtual void abort ()

  *Sets aborted to true.*

### Protected Attributes

- bool aborted = false

  *Defines if the process has been aborted, i.e.*

### 9.1.1 Detailed Description

A simple interface for classes that encapsulate abortable processes.
Definition at line 9 of file Abortable.h.

### 9.1.2 Member Function Documentation

#### 9.1.2.1 abort()

```
void Abortable::abort ( )  [virtual]
```
Sets aborted to *true*.
Reimplemented in Controller.
Definition at line 3 of file Abortable.cpp.
References aborted.
Referenced by Controller::abort().

### 9.1.3 Member Data Documentation

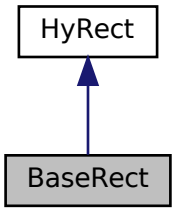
#### 9.1.3.1 aborted

```
bool Abortable::aborted = false  [protected]
```
Defines if the process has been aborted, i.e.
abort has been called.
Definition at line 14 of file Abortable.h.
Referenced by abort().
The documentation for this class was generated from the following files:

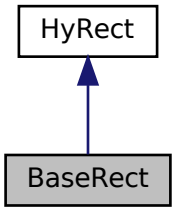
- /home/runner/work/simopticon/simopticon/src/utils/Abortable.h
- /home/runner/work/simopticon/simopticon/src/utils/Abortable.cpp

## 9.2 BaseRect Class Reference

A class representing a HyRect without a parent rectangle.
`#include "BaseRect.h"`
Inheritance diagram for BaseRect:



Collaboration diagram for BaseRect:




## Public Member Functions

- BaseRect (dimension D)

  *Creates a BaseRect representing a hypercube with the given dimensionality.*
- array< vector< dirCoordinate >, 2 > getSamplingVertices () override

  *Returns the coordinates of two opposite corner points of the rectangle.*

## Additional Inherited Members

### 9.2.1 Detailed Description

A class representing a HyRect without a parent rectangle.
This rectangle is always at the root of a partition tree and therefore has depth $t = 0$ and represents the whole search space.
Definition at line 12 of file BaseRect.h.

### 9.2.2 Constructor & Destructor Documentation

#### 9.2.2.1 BaseRect()

```
BaseRect::BaseRect (
            dimension D )  [explicit]
```

Creates a BaseRect representing a hypercube with the given dimensionality.

**Parameters**

| | |
|---|---|
| *D* | Number of dimensions of the search space. |

Definition at line 3 of file BaseRect.cpp.
References HyRect::HyRect().

### 9.2.3 Member Function Documentation

#### 9.2.3.1 getSamplingVertices()

```
array< vector< dirCoordinate >, 2 > BaseRect::getSamplingVertices ( )  [override], [virtual]
```

Returns the coordinates of two opposite corner points of the rectangle.

The returned vertices must be sampled. For BaseRect always returns one vector full of zeros and one vector full of ones.

**Returns**

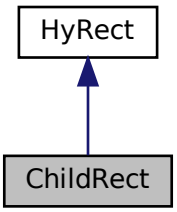
An array containing two dirCoordinate vectors of the sampled vertices.

Implements HyRect.
Definition at line 6 of file BaseRect.cpp.
References HyRect::D.
The documentation for this class was generated from the following files:

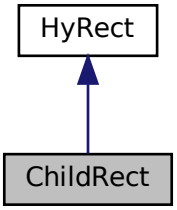
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/BaseRect.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/BaseRect.cpp

## 9.3 ChildRect Class Reference

A class representing a HyRect that has a parent HyRect.
`#include "ChildRect.h"`
Inheritance diagram for ChildRect:




Collaboration diagram for ChildRect:




### Public Member Functions

- ChildRect (position pos, shared_ptr< HyRect > parent)

  *Creates a ChildRect with the given relative position and parent rectangle.*
- array< vector< dirCoordinate >, 2 > getSamplingVertices () override

  *Returns the coordinates of two opposite corner points of the rectangle.*

- bool operator== (const HyRect &rect) const override

  *Checks if the current and the given HyRect objects are equal by comparing their pos, D, and t.*

## Private Attributes

- shared_ptr< HyRect > parent

  *Reference to the parent rectangle.*

## Additional Inherited Members

### 9.3.1 Detailed Description

A class representing a HyRect that has a parent HyRect.
Used for all HyRect where depth $t > 0$.
Definition at line 12 of file ChildRect.h.

### 9.3.2 Constructor & Destructor Documentation

#### 9.3.2.1 ChildRect()

```
ChildRect::ChildRect (
            position pos,
            shared_ptr< HyRect > parent )
```

Creates a ChildRect with the given relative position and parent rectangle.

**Parameters**

| | |
|---|---|
| *pos* | Relative position to the given parent rectangle. |
| *parent* | Parent rectangle in the partition tree. |

Definition at line 5 of file ChildRect.cpp.
References HyRect::HyRect(), HyRect::getD(), HyRect::getDepth(), and parent.
Referenced by HyRect::divide().

### 9.3.3 Member Function Documentation

#### 9.3.3.1 getSamplingVertices()

```
array< vector< dirCoordinate >, 2 > ChildRect::getSamplingVertices ( )  [override], [virtual]
```

Returns the coordinates of two opposite corner points of the rectangle.
The returned vertices must be sampled. The vertices are calculated recursively based on the sampling vertices of parent.

**Returns**

An array containing two dirCoordinate vectors of the sampled vertices.

Implements HyRect.
Definition at line 9 of file ChildRect.cpp.
References HyRect::getSamplingVertices(), HyRect::getSplitDim(), parent, and HyRect::pos.

#### 9.3.3.2 operator==()

```
bool ChildRect::operator== (
            const HyRect & rect ) const  [override], [virtual]
```

Checks if the current and the given HyRect objects are equal by comparing their pos, D, and t.

**Parameters**

| | |
|---|---|
| *rect* | HyRect to be compared. |

**Returns**

A boolean defining if the HyRect objects have the same position in the partition tree.

Reimplemented from HyRect.
Definition at line 27 of file ChildRect.cpp.
References HyRect::getPos(), parent, and HyRect::pos.

### 9.3.4 Member Data Documentation

#### 9.3.4.1 parent

```
shared_ptr<HyRect> ChildRect::parent  [private]
```

Reference to the parent rectangle.
Used for recursive calculation of getSamplingVertices.
Definition at line 17 of file ChildRect.h.
Referenced by ChildRect(), getSamplingVertices(), and operator==().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ChildRect.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ChildRect.cpp

## 9.4 CmpPairVectorSharedParameterFunctionvalue Struct Reference

This struct implements the comparison of two pairs of Parameter combination and function value.
`#include "ComparisonFunctions.h"`

### Public Member Functions

- bool operator() (const pair< vector< shared_ptr< Parameter >>, functionValue > &a, const pair< vector< shared_ptr< Parameter >>, functionValue > &b) const

  *Compares two pairs of Parameter combination and function value.*

### 9.4.1 Detailed Description

This struct implements the comparison of two pairs of Parameter combination and function value.
Definition at line 55 of file ComparisonFunctions.h.

### 9.4.2 Member Function Documentation

#### 9.4.2.1 operator()()

```
bool CmpPairVectorSharedParameterFunctionvalue::operator() (
            const pair< vector< shared_ptr< Parameter >>, functionValue > & a,
            const pair< vector< shared_ptr< Parameter >>, functionValue > & b ) const  [inline]
```

Compares two pairs of Parameter combination and function value.

**Parameters**

| | |
|---|---|
| *a* | First pair. |
| *b* | Second pair. |

**Returns**

Compares the function values. If they are the same, the Parameter combinations are compared.

Definition at line 62 of file ComparisonFunctions.h.
The documentation for this struct was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/ComparisonFunctions.h

# 9.5 CmpPtrFunctionvalue Struct Reference

This struct implements the comparison of two pointers to function values.
`#include "ComparisonFunctions.h"`

## Public Member Functions

- bool operator() (const functionValue ∗a, const functionValue ∗b) const

  *Compares two pointers to function values.*

## 9.5.1 Detailed Description

This struct implements the comparison of two pointers to function values.
Definition at line 42 of file ComparisonFunctions.h.

## 9.5.2 Member Function Documentation

### 9.5.2.1 operator()()

```
bool CmpPtrFunctionvalue::operator() (
            const functionValue * a,
            const functionValue * b ) const  [inline]
```

Compares two pointers to function values.

**Parameters**

| | |
|---|---|
| *a* | First pointer to a function value. |
| *b* | Second pointer to a function value. |

**Returns**

Compares ∗a and ∗b. If ∗a == ∗b the addresses are compared.

Definition at line 49 of file ComparisonFunctions.h.
The documentation for this struct was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/ComparisonFunctions.h

# 9.6 CmpSharedHyrect Struct Reference

This struct implements the comparison of two shared pointers to HyRect instances.
`#include "DirectComparisonFunctions.h"`

## Public Member Functions

- bool operator() (const shared_ptr< HyRect > &a, const shared_ptr< HyRect > &b) const

  *Compares two shared pointers to HyRect instances.*

### 9.6.1 Detailed Description

This struct implements the comparison of two shared pointers to HyRect instances.
Definition at line 17 of file DirectComparisonFunctions.h.

### 9.6.2 Member Function Documentation

#### 9.6.2.1 operator()()

```
bool CmpSharedHyrect::operator() (
            const shared_ptr< HyRect > & a,
            const shared_ptr< HyRect > & b ) const  [inline]
```

Compares two shared pointers to HyRect instances.

**Parameters**

| | |
|---|---|
| *a* | First pointer to a HyRect. |
| *b* | Second pointer to a HyRect. |

**Returns**

True if a has a lower HyRect::avgValue value than b. If both values are the same, compare the sampling vertices returned by HyRect::getSamplingVertices.

Definition at line 24 of file DirectComparisonFunctions.h.
References HyRect::getAvgValue(), and HyRect::getSamplingVertices().
The documentation for this struct was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectComparisonFunctions.h

## 9.7 CmpVectorSharedParameter Struct Reference

This struct implements the comparison of two vectors of Parameter references.
`#include "ComparisonFunctions.h"`

## Public Member Functions

- bool operator() (vector< shared_ptr< Parameter >> a, vector< shared_ptr< Parameter >> b) const

  *Compares two vectors of Parameter references.*

### 9.7.1 Detailed Description

This struct implements the comparison of two vectors of Parameter references.
Definition at line 19 of file ComparisonFunctions.h.

### 9.7.2 Member Function Documentation

#### 9.7.2.1 operator()()

```
bool CmpVectorSharedParameter::operator() (
            vector< shared_ptr< Parameter >> a,
            vector< shared_ptr< Parameter >> b ) const  [inline]
```

Compares two vectors of Parameter references.

**Parameters**

| | |
|---|---|
| *a* | First vector to be compared. |
| *b* | Second vector to be compared. |

**Returns**

True if a is smaller in size than b or if a is to be sorted before b by ascending order of coordinates.

Definition at line 26 of file ComparisonFunctions.h.
References Parameter::operator!=(), and Parameter::operator<().
The documentation for this struct was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/ComparisonFunctions.h

## 9.8 CommandLine Class Reference

A class containing functionality for executing commands on UNIX shell.
`#include "CommandLine.h"`

### Static Public Member Functions

- static unique_ptr< string > exec (string cmd)

  *Executes the given command in UNIX shell and returns the output (both stderr and stdout merged).*

### 9.8.1 Detailed Description

A class containing functionality for executing commands on UNIX shell.
Definition at line 18 of file CommandLine.h.

### 9.8.2 Member Function Documentation

#### 9.8.2.1 exec()

```
unique_ptr< string > CommandLine::exec (
            string cmd )  [static]
```

Executes the given command in UNIX shell and returns the output (both stderr and stdout merged).

**Parameters**

| | |
|---|---|
| *cmd* | Command to be executed. |

**Returns**

A string containing the output (sterr and stdout merged).

Definition at line 7 of file CommandLine.cpp.
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/utils/CommandLine.h

- /home/runner/work/simopticon/simopticon/src/utils/CommandLine.cpp

## 9.9 ConfigEditor Class Reference

A class capable of creating `.ini` files with certain options based on a complete `omnetpp.ini`.
`#include "ConfigEditor.h"`

### Public Member Functions

- ConfigEditor (filesystem::path directory, json controller)

  *Creates a ConfigEditor that creates config files in the given directory for simulation of the given controller.*
- void createConfig (const vector< shared_ptr< Parameter >> &params, size_t runNumber, unsigned int repeat)

  *Copies the config at CONFIG to a file `.tmpx.ini` where `x` is given by runNumber and edits the file for the purposes of the optimization.*
- void deleteConfig (size_t runId) const

  *Deletes the file `.tmpx.ini` from DIR where `x` is given by runId.*
- const filesystem::path & getDir () const

  *Returns the directory of the Plexe configuration.*
- filesystem::path getConfigPath (size_t runId) const

  *Returns the path to the created config for the Parameter combination with the given number.*
- filesystem::path getResultPath (size_t runId) const

  *Returns the path to the result files generated by simulating the Parameter combination with the given number.*

### Private Member Functions

- void setResultFiles (string &file, size_t runNumber)

  *Sets all output directories in the given file to a directory that is named after the given number and a subdirectory of RESULTS.*

### Static Private Member Functions

- static void replaceOption (string &file, string option, const string &value)

  *Replaces the value of the given key with the given new value in the given string.*
- static void replaceOption (string &file, string option, long value)

  *Replaces the value of the given key with the given new value in the given string.*
- static string getControllerOption (string &file)

  *Returns the key that defines the used controller in the given `.ini` file.*

### Private Attributes

- const filesystem::path DIR

  *Path to a directory containing a complete configuration of Plexe.*
- const filesystem::path CONFIG

  *Path to the `omnetpp.ini` file in DIR.*
- const filesystem::path RESULTS

  *Path to the `optResults` directory in DIR where the simulation result files are generated.*
- const json CONTROLLER

  *Configuration of the controller to be simulated.*

### 9.9.1 Detailed Description

A class capable of creating `.ini` files with certain options based on a complete `omnetpp.ini`.
Definition at line 23 of file ConfigEditor.h.

## 9.9.2 Constructor & Destructor Documentation

### 9.9.2.1 ConfigEditor()

```
ConfigEditor::ConfigEditor (
            filesystem::path directory,
            json controller )
```

Creates a ConfigEditor that creates config files in the given directory for simulation of the given controller.

**Parameters**

| | |
|---|---|
| *directory* | A path to the directory containing a Plexe configuration. |
| *controller* | A json object configuring the controller to be simulated. |

Definition at line 6 of file ConfigEditor.cpp.
References ConfigEditor().
Referenced by ConfigEditor().

## 9.9.3 Member Function Documentation

### 9.9.3.1 createConfig()

```
void ConfigEditor::createConfig (
            const vector< shared_ptr< Parameter >> & params,
            size_t runNumber,
            unsigned int repeat )
```

Copies the config at CONFIG to a file `.tmpx.ini` where `x` is given by *runNumber* and edits the file for the purposes of the optimization.
Sets the values of optimized parameters, controller, result directory and some options minimizing output of Plexe.

**Parameters**

| | |
|---|---|
| *params* | The Parameter combination to be simulated. |
| *runNumber* | An unique number of the simulated Parameter combination. |
| *repeat* | Number of repetitions to be simulated. |

Definition at line 11 of file ConfigEditor.cpp.
References setResultFiles().

### 9.9.3.2 deleteConfig()

```
void ConfigEditor::deleteConfig (
            size_t runId ) const
```

Deletes the file `.tmpx.ini` from DIR where `x` is given by *runId*.

**Parameters**

| | |
|---|---|
| *runId* | Number of the configuration file to be deleted. |

Definition at line 90 of file ConfigEditor.cpp.

#### 9.9.3.3 getConfigPath()

```
filesystem::path ConfigEditor::getConfigPath (
            size_t runId ) const
```

Returns the path to the created config for the Parameter combination with the given number.

**Parameters**

| | |
|---|---|
| *runId* | Number of the Parameter combination. |

**Returns**

A path to the config for the given runId.

Definition at line 80 of file ConfigEditor.cpp.

#### 9.9.3.4 getControllerOption()

```
string ConfigEditor::getControllerOption (
            string & file ) [static], [private]
```

Returns the key that defines the used controller in the given `.ini` file.
That is necessary for backwards compatability reasons because said key changed in Plexe 3.1.

**Parameters**

| | |
|---|---|
| *file* | A string containing the contents of an `.ini` file. |

**Returns**

A string containing the key where the used controller is defined.

Definition at line 69 of file ConfigEditor.cpp.

#### 9.9.3.5 getDir()

```
const filesystem::path & ConfigEditor::getDir ( ) const
```

Returns the directory of the Plexe configuration.

**Returns**

The path stored in DIR

Definition at line 94 of file ConfigEditor.cpp.

#### 9.9.3.6 getResultPath()

```
filesystem::path ConfigEditor::getResultPath (
            size_t runId ) const
```

Returns the path to the result files generated by simulating the Parameter combination with the given number.

**Parameters**

| | |
|---|---|
| *runId* | Number of the Parameter combination. |

**Returns**

A path to the result files for the given runId.

Definition at line 85 of file ConfigEditor.cpp.

#### 9.9.3.7 replaceOption() [1/2]

```
void ConfigEditor::replaceOption (
            string & file,
            string option,
            const string & value )  [static], [private]
```

Replaces the value of the given key with the given new value in the given string.

**Parameters**

| | |
|---|---|
| *file* | A string containing the contents of an `.ini` file. |
| *option* | A string representing a key in the given file. |
| *value* | The new value of the given option in the given file. |

Definition at line 42 of file ConfigEditor.cpp.

#### 9.9.3.8 replaceOption() [2/2]

```
void ConfigEditor::replaceOption (
            string & file,
            string option,
            long value )  [static], [private]
```

Replaces the value of the given key with the given new value in the given string.
Basically parses the given value to string and calls replaceOption(string &, string, const string &).

**Parameters**

| | |
|---|---|
| *file* | A string containing the contents of an `.ini` file. |
| *option* | A string representing a key in the given file. |
| *value* | The new value of the given option in the given file. |

Definition at line 56 of file ConfigEditor.cpp.

#### 9.9.3.9 setResultFiles()

```
void ConfigEditor::setResultFiles (
            string & file,
            size_t runNumber )  [private]
```

Sets all output directories in the given file to a directory that is named after the given number and a subdirectory of RESULTS.

**Parameters**

| | |
|---|---|
| *file* | A string containing the contents of an `.ini` file. |
| *runNumber* | The unique number of the Parameter combination. |

Definition at line 60 of file ConfigEditor.cpp.
Referenced by createConfig().

### 9.9.4 Member Data Documentation

#### 9.9.4.1 CONFIG

`const filesystem::path ConfigEditor::CONFIG [private]`
Path to the `omnetpp.ini` file in DIR.
Definition at line 33 of file ConfigEditor.h.

#### 9.9.4.2 CONTROLLER

`const json ConfigEditor::CONTROLLER [private]`
Configuration of the controller to be simulated.
Can be set in config.
Definition at line 42 of file ConfigEditor.h.

#### 9.9.4.3 DIR

`const filesystem::path ConfigEditor::DIR [private]`
Path to a directory containing a complete configuration of Plexe.
Can be set in config.
Definition at line 29 of file ConfigEditor.h.

#### 9.9.4.4 RESULTS

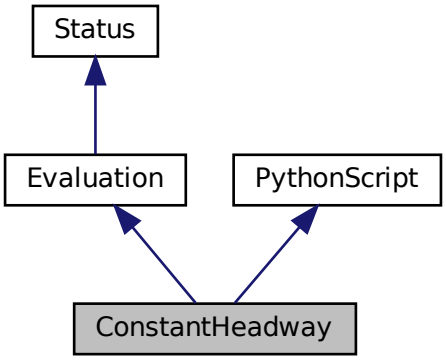
`const filesystem::path ConfigEditor::RESULTS [private]`
Path to the `optResults` directory in DIR where the simulation result files are generated.
Definition at line 37 of file ConfigEditor.h.
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/runner/plexe/ConfigEditor.h
- /home/runner/work/simopticon/simopticon/src/runner/plexe/ConfigEditor.cpp

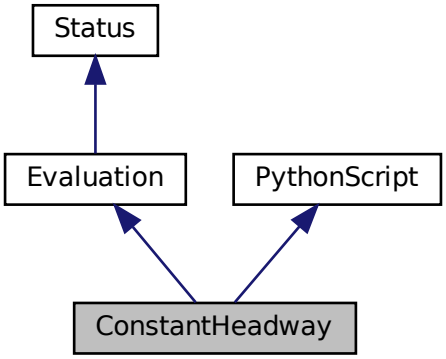
## 9.10 ConstantHeadway Class Reference

A wrapper for the constant_headway.py script.
`#include "ConstantHeadway.h"`

Inheritance diagram for ConstantHeadway:




Collaboration diagram for ConstantHeadway:




## Public Member Functions

- ConstantHeadway (unsigned int nrThreads, const filesystem::path &pathToScript)

  *Creates a ConstantHeadway object that uses no more than the given number of threads and interfaces with the multithreaded function of the given script.*
- functionValue processOutput (filesystem::path path, set< runId > experimentIds) override

  *Returns a value to the results of a single simulation run.*
- map< pair< filesystem::path, set< runId > >, functionValue > processOutput (const set< pair< filesystem::path, set< runId >>> &experimentResults) override

  *Returns values to the results of multiple simulation runs.*
- string getName () override

  *Returns a string representing the name of the implementing component in natural language.*
- string getStatus () override

  *Returns a string representing the current state of the implementing component.*

- string getStatusBar () override

  *Returns a string representing the current progress of the calculations of the implementing component.*

## Private Member Functions

- PyObject * secureValue (PyObject *object)

  *Helper function checking if the given object is a null-pointer.*

## Private Attributes

- const unsigned int NR_THREADS

  *Maximum number of threads to use for concurrent evaluation.*
- unsigned int usedThreads = 0

  *Number of threads currently used for concurrent evaluation.*

## Additional Inherited Members

### 9.10.1 Detailed Description

A wrapper for the constant_headway.py script.
Definition at line 25 of file ConstantHeadway.h.

### 9.10.2 Constructor & Destructor Documentation

#### 9.10.2.1 ConstantHeadway()

```
ConstantHeadway::ConstantHeadway (
            unsigned int nrThreads,
            const filesystem::path & pathToScript )
```

Creates a ConstantHeadway object that uses no more than the given number of threads and interfaces with the multithreaded function of the given script.

**Parameters**

| | |
|---|---|
| *nrThreads* | Maximum number of threads used for concurrent calculations. |
| *pathToScript* | Path to the constant_headway.py script. |

Definition at line 8 of file ConstantHeadway.cpp.
References ConstantHeadway(), and NR_THREADS.
Referenced by ConstantHeadway().

### 9.10.3 Member Function Documentation

#### 9.10.3.1 getName()

```
string ConstantHeadway::getName ( )  [override], [virtual]
```

Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented from Status.
Definition at line 72 of file ConstantHeadway.cpp.

#### 9.10.3.2 getStatus()

```
string ConstantHeadway::getStatus ( )  [override], [virtual]
```

Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented from Status.
Definition at line 76 of file ConstantHeadway.cpp.
References NR_THREADS.

#### 9.10.3.3 getStatusBar()

```
string ConstantHeadway::getStatusBar ( )  [override], [virtual]
```

Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented from Status.
Definition at line 80 of file ConstantHeadway.cpp.
References NR_THREADS, and usedThreads.

#### 9.10.3.4 processOutput() [1/2]

```
map< pair< filesystem::path, set< runId > >, functionValue > ConstantHeadway::processOutput
(
            const set< pair< filesystem::path, set< runId >>> & experimentResults )  [override],
[virtual]
```

Returns values to the results of multiple simulation runs.
Passes given parameters to the multithreaded function of constant_headway.py.

**Parameters**

| | |
|---|---|
| *experimentResults* | Paths to and identifiers of the simulation results. |

**Returns**

A map which maps the given results to their respective performance value.

Reimplemented from Evaluation.
Definition at line 13 of file ConstantHeadway.cpp.

#### 9.10.3.5 processOutput() [2/2]

```
functionValue ConstantHeadway::processOutput (
            filesystem::path path,
            set< runId > experimentIds )  [override], [virtual]
```

Returns a value to the results of a single simulation run.
Basically calls processOutput(const set<pair<filesystem::path, set<runId>>> &) with the given values.

**Parameters**

| | |
|---|---|
| *path* | Path to the result files. |
| *experimentIds* | Identifiers of certain simulation runs within the directory represented by the given path. |

**Returns**

A value that represents the performance of the simulation - the lower the better.

Implements Evaluation.
Definition at line 58 of file ConstantHeadway.cpp.

#### 9.10.3.6 secureValue()

```
PyObject * ConstantHeadway::secureValue (
            PyObject * object )  [private]
```

Helper function checking if the given object is a null-pointer.
If so the constant_headway.py script is disconnected and an error is thrown.

**Parameters**

| | |
|---|---|
| *object* | Pointer to PyObject that must be tested. |

**Returns**

The given pointer, if no error was thrown.

Definition at line 62 of file ConstantHeadway.cpp.

### 9.10.4 Member Data Documentation

#### 9.10.4.1 NR_THREADS

```
const unsigned int ConstantHeadway::NR_THREADS  [private]
```

Maximum number of threads to use for concurrent evaluation.
Can be set in config.
Definition at line 31 of file ConstantHeadway.h.
Referenced by ConstantHeadway(), getStatus(), and getStatusBar().

#### 9.10.4.2 usedThreads

```
unsigned int ConstantHeadway::usedThreads = 0  [private]
```

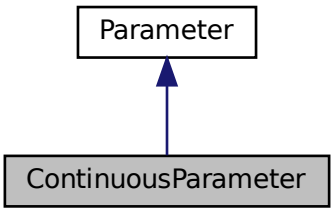
Number of threads currently used for concurrent evaluation.
Used in getStatusBar.
Definition at line 36 of file ConstantHeadway.h.
Referenced by getStatusBar().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/ConstantHeadway.h
- /home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/ConstantHeadway.cpp

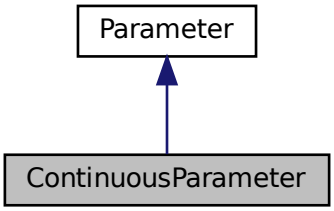
## 9.11 ContinuousParameter Class Reference

Implements a Parameter using continuos values in the form of floating point numbers.

`#include "ContinuousParameter.h"`
Inheritance diagram for ContinuousParameter:




Collaboration diagram for ContinuousParameter:




## Public Member Functions

- ContinuousParameter (shared_ptr< ParameterDefinition > def, coordinate value)

  *Creates a ContinuousParameter with the given ParameterDefinition and value.*
- ContinuousParameter (shared_ptr< ParameterDefinition > def)

  *Creates a ContinuousParameter with the given ParameterDefinition and the initial value being the mean between minimum and maximum.*
- coordinate getVal () const override

  *Returns the current value of val.*
- void setVal (coordinate val) override

  *Sets the value of val to the given value.*

## Private Attributes

- coordinate val

  *Value of the ContinuousParameter.*

### 9.11.1 Detailed Description

Implements a Parameter using continuos values in the form of floating point numbers.
Definition at line 11 of file ContinuousParameter.h.

## 9.11.2 Constructor & Destructor Documentation

### 9.11.2.1 ContinuousParameter() [1/2]

```
ContinuousParameter::ContinuousParameter (
            shared_ptr< ParameterDefinition > def,
            coordinate value )
```

Creates a ContinuousParameter with the given ParameterDefinition and value.

- Checks if given value is in bounds set by the ParameterDefinition.

**Parameters**

| | |
|---|---|
| *def* | ParameterDefinition of the Parameter. |
| *value* | Initial value of the Parameter. |

Definition at line 6 of file ContinuousParameter.cpp.
References Parameter::Parameter(), Parameter::getMax(), Parameter::getMin(), and val.
Referenced by ContinuousParameter(), and ParameterNormalizer::denormalize().

### 9.11.2.2 ContinuousParameter() [2/2]

```
ContinuousParameter::ContinuousParameter (
            shared_ptr< ParameterDefinition > def ) [explicit]
```

Creates a ContinuousParameter with the given ParameterDefinition and the initial value being the mean between minimum and maximum.

**Parameters**

| | |
|---|---|
| *def* | ParameterDefinition of the Parameter. |

Definition at line 14 of file ContinuousParameter.cpp.
References ContinuousParameter(), Parameter::getMax(), and Parameter::getMin().

## 9.11.3 Member Function Documentation

### 9.11.3.1 getVal()

```
coordinate ContinuousParameter::getVal ( ) const [override], [virtual]
```

Returns the current value of val.

**Returns**

A coordinate representing the value of the ContinuousParameter.

Implements Parameter.
Definition at line 19 of file ContinuousParameter.cpp.
References val.

### 9.11.3.2 setVal()

```
void ContinuousParameter::setVal (
            coordinate val ) [override], [virtual]
```

Sets the value of val to the given value.

Checks if given value is in bounds set by the ParameterDefinition.

**Parameters**

| | |
|---|---|
| *val* | Value to set the ContinuousParameter to. |

Implements Parameter.
Definition at line 23 of file ContinuousParameter.cpp.
References Parameter::getMax(), Parameter::getMin(), and val.

### 9.11.4 Member Data Documentation

#### 9.11.4.1 val

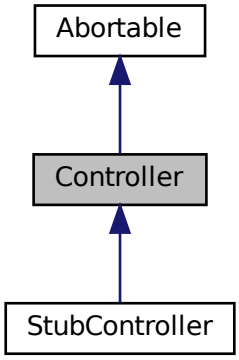
```
coordinate ContinuousParameter::val  [private]
```

Value of the ContinuousParameter.
Definition at line 16 of file ContinuousParameter.h.
Referenced by ContinuousParameter(), getVal(), and setVal().
The documentation for this class was generated from the following files:

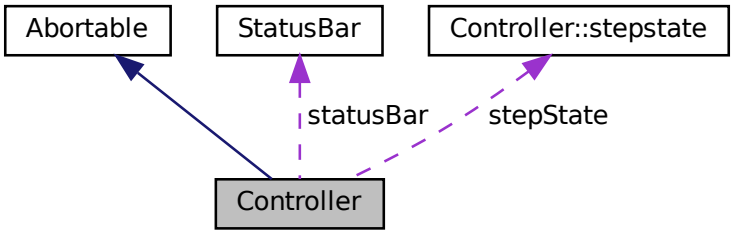
- /home/runner/work/simopticon/simopticon/src/parameters/ContinuousParameter.h
- /home/runner/work/simopticon/simopticon/src/parameters/ContinuousParameter.cpp

## 9.12 Controller Class Reference

A class responsible for communication between Optimizer, SimulationRunner and Evaluation and also user interaction such as tracking results, updating StatusBar and handling interrupts by the user via Abortable.
`#include "Controller.h"`
Inheritance diagram for Controller:



Collaboration diagram for Controller:




## Classes

- struct stepstate

  *A struct keeping track of the currently running optimization step for StatusBar::updateStatus.*

## Public Member Functions

- Controller (const filesystem::path &configPath)

  *Creates a Controller which uses Optimizer, SimulationRunner and Evaluation as specified in the given config files.*
- void run ()

  *Starts optimization process by calling Optimizer::runOptimization.*
- map< vector< shared_ptr< Parameter > >, functionValue > requestValues (const list< vector< shared_ptr< Parameter >>> &params)

  *Searches valueMap for results to given Parameter combinations.*
- ValueMap & getValueMap ()

  *Returns valueMap.*
- void abort () override

  *Aborts optimizer using Optimizer::abort.*

## Protected Attributes

- StatusBar statusBar

  *StatusBar object used for output.*
- unique_ptr< Optimizer > optimizer

  *Optimizer defining an optimization strategy.*
- unique_ptr< SimulationRunner > runner

  *SimulationRunner able to run simulations with certain Parameter combinations.*
- unique_ptr< Evaluation > evaluation

  *Evaluation capable of evaluating data produced by runner.*
- unique_ptr< ValueMap > valueMap

  *ValueMap containing all values gathered by simulating and evaluating certain Parameter combinations.*

## Private Member Functions

- virtual map< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, CmpVectorSharedParameter > runSimulations (const set< vector< shared_ptr< Parameter >>, CmpVectorSharedParameter > &runs)

*Calls the runner to run simulations for the given Parameter combinations.*

- virtual map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > evaluate (const map< vector< shared_ptr< Parameter >>, pair< filesystem::path, set< runId >>, CmpVectorSharedParameter > &simulationResults)

  *Calls the evaluation to evaluate the given result files.*
- virtual void removeOldResultfiles ()

  *Removes all result files that don't belong to the best n results, where n is configured in main config.*
- virtual void updateStatus ()

  *Updates the statusBar using StatusBar::updateStatus.*

## Private Attributes

- bool keepFiles

  *Defines if result files of best simulations are kept after optimization.*
- map< vector< shared_ptr< Parameter > >, filesystem::path > topResults

  *Saves the best n Parameter combinations and the corresponding path to the result files, if keepFiles is true.*
- struct Controller::stepstate stepState

  *An object keeping track of the current optimization step.*
- chrono::milliseconds statusInterval = chrono::milliseconds(0)

  *Interval of updates of StatusBar using updateStatus in concurrent status thread.*

### 9.12.1 Detailed Description

A class responsible for communication between Optimizer, SimulationRunner and Evaluation and also user interaction such as tracking results, updating StatusBar and handling interrupts by the user via Abortable.
Definition at line 36 of file Controller.h.

### 9.12.2 Constructor & Destructor Documentation

#### 9.12.2.1 Controller()

```
Controller::Controller (
            const filesystem::path & configPath )  [explicit]
```

Creates a Controller which uses Optimizer, SimulationRunner and Evaluation as specified in the given config files.

**Parameters**

| | |
|---|---|
| *configPath* | Path to the main config. Chosen by first command line argument. |

Definition at line 27 of file Controller.cpp.
References statusInterval, and valueMap.

### 9.12.3 Member Function Documentation

#### 9.12.3.1 abort()

```
void Controller::abort ( )  [override], [virtual]
```

Aborts optimizer using Optimizer::abort.
Aborts the concurrent thread that regularly updates statusBar.
Reimplemented from Abortable.
Definition at line 218 of file Controller.cpp.
References Abortable::abort(), and optimizer.

#### 9.12.3.2 evaluate()

```
map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > Controller↩
::evaluate (
            const map< vector< shared_ptr< Parameter >>, pair< filesystem::path, set< runId
>>, CmpVectorSharedParameter > & simulationResults )  [private], [virtual]
```

Calls the evaluation to evaluate the given result files.
Updates statusBar before and after execution of evaluation.

**Parameters**

| | |
|---|---|
| *simulationResults* | A map which maps the Parameter combinations that must be evaluated to their respective file paths of simulation results and runIds. |

**Returns**

A map which maps the given Parameter combinations to their respective functionValue.

Reimplemented in StubController.
Definition at line 176 of file Controller.cpp.
References updateStatus().

#### 9.12.3.3 getValueMap()

```
ValueMap & Controller::getValueMap ( )
```

Returns valueMap.

**Returns**

A ValueMap object.

Definition at line 144 of file Controller.cpp.
References valueMap.
Referenced by Optimizer::getValueMap().

#### 9.12.3.4 removeOldResultfiles()

```
void Controller::removeOldResultfiles ( )  [private], [virtual]
```

Removes all result files that don't belong to the best *n* results, where *n* is configured in main config.
If keepFiles is *false*, all result files are removed.
Reimplemented in StubController.
Definition at line 192 of file Controller.cpp.
Referenced by requestValues().

#### 9.12.3.5 requestValues()

```
map< vector< shared_ptr< Parameter > >, functionValue > Controller::requestValues (
            const list< vector< shared_ptr< Parameter >>> & params )
```

Searches valueMap for results to given Parameter combinations.
Each combination that hasn't been simulated is simulated and evaluated using runSimulations and evaluate. Updates statusBar before and after execution.

**Parameters**

| | |
|---|---|
| *params* | A set of Parameter combinations to be evaluated. |

**Returns**

A map which maps the given Parameter combinations to their respective functionValue.

Definition at line 110 of file Controller.cpp.
References Controller::stepstate::next(), removeOldResultfiles(), and updateStatus().
Referenced by Optimizer::requestValues().

#### 9.12.3.6 run()

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

Starts optimization process by calling Optimizer::runOptimization.
Creates concurrent thread that updates statusBar every statusInterval milliseconds. Prints results in command line after optimization is done using StatusBar::printResults.
Definition at line 148 of file Controller.cpp.
References Controller::stepstate::next(), optimizer, Optimizer::runOptimization(), statusInterval, and updateStatus().

#### 9.12.3.7 runSimulations()

```
map< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, CmpVectorSharedParameter
> Controller::runSimulations (
            const set< vector< shared_ptr< Parameter >>, CmpVectorSharedParameter > & runs
)  [private], [virtual]
```

Calls the runner to run simulations for the given Parameter combinations.
Updates statusBar before and after execution of simulations.

**Parameters**

| | |
|---|---|
| *runs* | A set of Parameter combinations to be executed. |

**Returns**

A map which maps the given Parameter combinations to their respective result file paths and runIds.

Reimplemented in StubController.
Definition at line 169 of file Controller.cpp.

#### 9.12.3.8 updateStatus()

```
void Controller::updateStatus ( )  [private], [virtual]
```

Updates the statusBar using StatusBar::updateStatus.
Definition at line 210 of file Controller.cpp.
References Controller::stepstate::stepChanged.
Referenced by evaluate(), requestValues(), and run().

### 9.12.4 Member Data Documentation

#### 9.12.4.1 evaluation

```
unique_ptr<Evaluation> Controller::evaluation  [protected]
```

Evaluation capable of evaluating data produced by runner.
Definition at line 133 of file Controller.h.

#### 9.12.4.2 keepFiles

```
bool Controller::keepFiles  [private]
```

Defines if result files of best simulations are kept after optimization.
Can be set in main config.
Definition at line 41 of file Controller.h.

#### 9.12.4.3 optimizer

```
unique_ptr<Optimizer> Controller::optimizer  [protected]
```

Optimizer defining an optimization strategy.
Definition at line 125 of file Controller.h.
Referenced by abort(), and run().

#### 9.12.4.4 runner

```
unique_ptr<SimulationRunner> Controller::runner  [protected]
```

SimulationRunner able to run simulations with certain Parameter combinations.
Definition at line 129 of file Controller.h.

#### 9.12.4.5 statusBar

```
StatusBar Controller::statusBar  [protected]
```

StatusBar object used for output.
Definition at line 121 of file Controller.h.

#### 9.12.4.6 statusInterval

```
chrono::milliseconds Controller::statusInterval = chrono::milliseconds(0)  [private]
```

Interval of updates of StatusBar using updateStatus in concurrent status thread.
Definition at line 86 of file Controller.h.
Referenced by Controller(), and run().

#### 9.12.4.7 topResults

```
map<vector<shared_ptr<Parameter> >, filesystem::path> Controller::topResults  [private]
```

Saves the best *n* Parameter combinations and the corresponding path to the result files, if keepFiles is true.
*n* can be set in main config.
Definition at line 45 of file Controller.h.

#### 9.12.4.8 valueMap

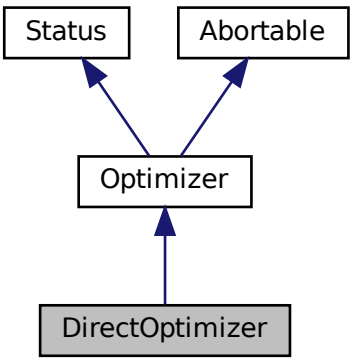
```
unique_ptr<ValueMap> Controller::valueMap  [protected]
```

ValueMap containing all values gathered by simulating and evaluating certain Parameter combinations.
Definition at line 137 of file Controller.h.
Referenced by Controller(), and getValueMap().
The documentation for this class was generated from the following files:

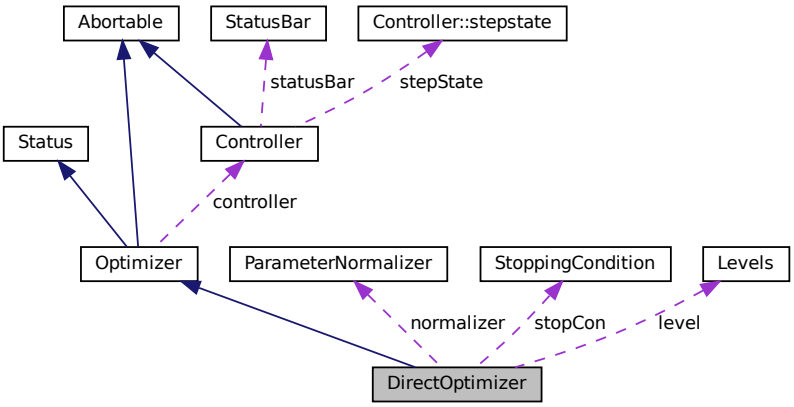
- /home/runner/work/simopticon/simopticon/src/controller/Controller.h
- /home/runner/work/simopticon/simopticon/src/controller/Controller.cpp

## 9.13 DirectOptimizer Class Reference

A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.
`#include "DirectOptimizer.h"`
Inheritance diagram for DirectOptimizer:




Collaboration diagram for DirectOptimizer:




### Public Member Functions

- DirectOptimizer (Controller &ctrl, const list< shared_ptr< ParameterDefinition >> &params, StoppingCondition con, bool trackProgress, bool printValues)

  *Creates a DirectOptimizer that evaluates functions with the given Controller, optimizes the given ParameterDefinition list and stops as defined by the given StoppingCondition.*
- void runOptimization () override

  *Starts the optimization using the DIRECT algorithm.*
- string getName () override

  *Returns a string representing the name of the implementing component in natural language.*
- string getStatus () override

*Returns a string representing the current state of the implementing component.*

- string getStatusBar () override

  *Returns a string representing the current progress of the calculations of the implementing component.*

- size_t getPartitionSize ()

  *Returns the number of rectangles stored in activeRects.*

## Private Member Functions

- map< vector< dirCoordinate >, functionValue > getValues (const list< vector< dirCoordinate >> &points)

  *Returns the function values at the given points.*

- list< shared_ptr< HyRect > > optimalRectangles (size_t nrRects, functionValue phi)

  *Finds potentially optimal rectangles that should be divided in the current iteration.*

- void addActiveRects (const list< shared_ptr< HyRect >> &rects)

  *Requests values at the corners of the given rectangles and add all given HyRect instances to activeRects.*

- void removeActiveRects (const list< shared_ptr< HyRect >> &rects)

  *Removes the given rectangles from activeRects.*

- void saveProgress (functionValue bestVal, size_t evaluations, size_t nrRects) const

  *Prints the current number of iterations, evaluations, rectangles and the current optimal value to a `.csv` file.*

- void saveValues ()

  *Prints all evaluated Parameter combinations and their respective values to a `.csv` file.*

## Static Private Member Functions

- static functionValue estimatedValue (const shared_ptr< HyRect > &rect, double k)

  *Calculates the minimum expected value in a rectangle when the given Lipschitz constant is assumed.*

## Private Attributes

- const dimension D

  *Number of parameters to be optimized (meaning dimensions of the search space).*

- size_t iterations = 0

  *Number of iterations completed.*

- StoppingCondition stopCon

  *An object deciding when the optimization stops.*

- Levels level

  *An object used switching between different levels between global and local search.*

- ParameterNormalizer normalizer

  *An object used for transformation between the unit hypercube used in DIRECT and the actual parameter space.*

- bool trackProgress

  *Defines if the current number of iterations, evaluations, rectangles and the optimal value should be recorded into a `.csv` file after each iteration.*

- bool printValues

  *Defines if all found values should be recorded in a `.csv` file after optimization has finished.*

- map< depth, set< shared_ptr< HyRect >, CmpSharedHyrect >, greater<> > activeRects

  *Holds all rectangles that are immediate part of the current partition.*

## Additional Inherited Members

### 9.13.1 Detailed Description

A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.
Definition at line 32 of file DirectOptimizer.h.

## 9.13.2 Constructor & Destructor Documentation

### 9.13.2.1 DirectOptimizer()

```
DirectOptimizer::DirectOptimizer (
            Controller & ctrl,
            const list< shared_ptr< ParameterDefinition >> & params,
            StoppingCondition con,
            bool trackProgress,
            bool printValues )
```

Creates a DirectOptimizer that evaluates functions with the given Controller, optimizes the given ParameterDefinition list and stops as defined by the given StoppingCondition.

**Parameters**

| | |
|---|---|
| *ctrl* | Controller to be used for evaluating the optimized function. |
| *params* | ParameterDefinition list to be optimized. |
| *con* | StoppingCondition defining the end of optimization. |
| *trackProgress* | Defines whether the progress should be printed in a `.csv` file. |
| *printValues* | Defines whether all obtained values should be printed in a `.csv` file after optimization. |

Definition at line 11 of file DirectOptimizer.cpp.
References DirectOptimizer(), level, printValues, and trackProgress.
Referenced by DirectOptimizer().

## 9.13.3 Member Function Documentation

### 9.13.3.1 addActiveRects()

```
void DirectOptimizer::addActiveRects (
            const list< shared_ptr< HyRect >> & rects )  [private]
```

Requests values at the corners of the given rectangles and add all given HyRect instances to activeRects.

**Parameters**

| | |
|---|---|
| *rects* | Rectangles to be evaluated and added. |

Definition at line 106 of file DirectOptimizer.cpp.

### 9.13.3.2 estimatedValue()

```
functionValue DirectOptimizer::estimatedValue (
            const shared_ptr< HyRect > & rect,
            double k )  [static], [private]
```

Calculates the minimum expected value in a rectangle when the given Lipschitz constant is assumed.

**Parameters**

| | |
|---|---|
| *rect* | Rectangle the minimum is searched for. |
| *k* | Lipschitz constant that is assumed in this rectangle. |

**Returns**

A value representing an estimation of the absolute minimum reachable in this rectangle.

Definition at line 86 of file DirectOptimizer.cpp.
References HyRect::getAvgValue(), and HyRect::getDiagonalLength().

#### 9.13.3.3 getName()

`string DirectOptimizer::getName ( ) [override], [virtual]`
Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented from Status.
Definition at line 189 of file DirectOptimizer.cpp.

#### 9.13.3.4 getPartitionSize()

`size_t DirectOptimizer::getPartitionSize ( )`
Returns the number of rectangles stored in activeRects.

**Returns**

A number representing the size of the partition.

Definition at line 206 of file DirectOptimizer.cpp.
Referenced by getStatus().

#### 9.13.3.5 getStatus()

`string DirectOptimizer::getStatus ( ) [override], [virtual]`
Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented from Status.
Definition at line 193 of file DirectOptimizer.cpp.
References Levels::getLevel(), getPartitionSize(), iterations, and level.

#### 9.13.3.6 getStatusBar()

`string DirectOptimizer::getStatusBar ( ) [override], [virtual]`
Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented from Status.
Definition at line 202 of file DirectOptimizer.cpp.

#### 9.13.3.7 getValues()

```
map< vector< dirCoordinate >, functionValue > DirectOptimizer::getValues (
            const list< vector< dirCoordinate >> & points )  [private]
```

Returns the function values at the given points.
Basically transforms the given points from dirCoordinates in the hypercube to actual coordinates in the parameter space using normalizer and calls requestValues.

**Parameters**

| | |
|---|---|
| *points* | List of points in the hypercube to be evaluated. |

**Returns**

A map which maps the given points to their respective values.

Definition at line 68 of file DirectOptimizer.cpp.

#### 9.13.3.8 optimalRectangles()

```
list< shared_ptr< HyRect > > DirectOptimizer::optimalRectangles (
            size_t nrRects,
            functionValue phi )  [private]
```

Finds potentially optimal rectangles that should be divided in the current iteration.
First filters for only the best rectangles of a size from a subset of all activeRects determined by level. Then uses GrahamScan to filter after the first condition of the DIRECT algorithm. Finally filters for the second condition of the DIRECT algorithm.

**Parameters**

| | |
|---|---|
| *nrRects* | Size of the partition (meaning number of rectangles in activeRects). |
| *phi* | Value at the current minimum. |

**Returns**

A list of potentially optimal rectangles.

Definition at line 90 of file DirectOptimizer.cpp.

#### 9.13.3.9 removeActiveRects()

```
void DirectOptimizer::removeActiveRects (
            const list< shared_ptr< HyRect >> & rects )  [private]
```

Removes the given rectangles from activeRects.

**Parameters**

| | |
|---|---|
| *rects* | Rectangles to be removed. |

Definition at line 141 of file DirectOptimizer.cpp.

#### 9.13.3.10 runOptimization()

```
void DirectOptimizer::runOptimization ( )  [override], [virtual]
```

Starts the optimization using the DIRECT algorithm.

Only returns when an iteration has completed and stopCon deems the optimization complete or when abort was called in the last iteration.
Implements Optimizer.
Definition at line 18 of file DirectOptimizer.cpp.
References Optimizer::getValueMap(), Levels::isGlobal(), iterations, Levels::L3_EPSILON, level, Levels::nextLevel(), printValues, saveProgress(), saveValues(), Levels::setGlobal(), and trackProgress.

#### 9.13.3.11 saveProgress()

```
void DirectOptimizer::saveProgress (
            functionValue bestVal,
            size_t evaluations,
            size_t nrRects ) const  [private]
```

Prints the current number of iterations, evaluations, rectangles and the current optimal value to a `.csv` file.

**Parameters**

| | |
|---|---|
| *bestVal* | Value at the current minimum. |
| *evaluations* | Number of evaluations conducted by the optimization. |
| *nrRects* | Number of rectangles in the current partition (meaning number of rectangles in activeRects). |

Definition at line 152 of file DirectOptimizer.cpp.
Referenced by runOptimization().

#### 9.13.3.12 saveValues()

```
void DirectOptimizer::saveValues ( )  [private]
```

Prints all evaluated Parameter combinations and their respective values to a `.csv` file.
Definition at line 164 of file DirectOptimizer.cpp.
Referenced by runOptimization().

### 9.13.4 Member Data Documentation

#### 9.13.4.1 activeRects

```
map<depth, set<shared_ptr<HyRect>, CmpSharedHyrect>, greater<> > DirectOptimizer::activeRects  [private]
```

Holds all rectangles that are immediate part of the current partition.
This includes all rectangles which have not been divided yet. They are grouped by HyRect::t and sorted by HyRect::avgValue which simplifies the search for potentially optimal rectangles in optimalRectangles.
Definition at line 70 of file DirectOptimizer.h.

#### 9.13.4.2 D

```
const dimension DirectOptimizer::D  [private]
```

Number of parameters to be optimized (meaning dimensions of the search space).
Definition at line 37 of file DirectOptimizer.h.

#### 9.13.4.3 iterations

```
size_t DirectOptimizer::iterations = 0  [private]
```

Number of iterations completed.

Definition at line 41 of file DirectOptimizer.h.
Referenced by getStatus(), and runOptimization().

#### 9.13.4.4 level

`Levels DirectOptimizer::level [private]`
An object used switching between different levels between global and local search.
Definition at line 49 of file DirectOptimizer.h.
Referenced by DirectOptimizer(), getStatus(), and runOptimization().

#### 9.13.4.5 normalizer

`ParameterNormalizer DirectOptimizer::normalizer [private]`
An object used for transformation between the unit hypercube used in DIRECT and the actual parameter space.
Definition at line 53 of file DirectOptimizer.h.

#### 9.13.4.6 printValues

`bool DirectOptimizer::printValues [private]`
Defines if all found values should be recorded in a `.csv` file after optimization has finished.
Can be set in config.
Definition at line 63 of file DirectOptimizer.h.
Referenced by DirectOptimizer(), and runOptimization().

#### 9.13.4.7 stopCon

`StoppingCondition DirectOptimizer::stopCon [private]`
An object deciding when the optimization stops.
Definition at line 45 of file DirectOptimizer.h.

#### 9.13.4.8 trackProgress

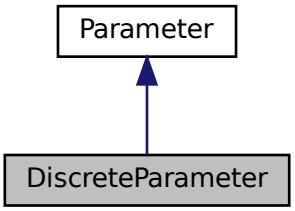
`bool DirectOptimizer::trackProgress [private]`
Defines if the current number of iterations, evaluations, rectangles and the optimal value should be recorded into a `.csv` file after each iteration.
Can be set in config.
Definition at line 58 of file DirectOptimizer.h.
Referenced by DirectOptimizer(), and runOptimization().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectOptimizer.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectOptimizer.cpp

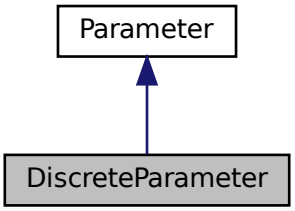
## 9.14 DiscreteParameter Class Reference

Implements a Parameter using discrete values.
`#include "DiscreteParameter.h"`

Inheritance diagram for DiscreteParameter:




Collaboration diagram for DiscreteParameter:




## Public Member Functions

- DiscreteParameter (shared_ptr< ParameterDefinition > def, double step, double value)

  *Creates a DiscreteParameter with the given ParameterDefinition, step and value.*
- DiscreteParameter (shared_ptr< ParameterDefinition > def, double step)

  *Creates a DiscreteParameter with the given ParameterDefinition and step.*
- int getTimes () const

  *Returns the value of times.*
- void setTimes (int newTimes)

  *Sets the value of times to the given value.*
- double getStep () const

  *Returns the value of step.*
- double getOffset () const

  *Returns the value of offset.*
- coordinate getVal () const override

  *Returns the current value of the DiscreteParameter as calculated by the following formula:* $val = times \cdot step + offset$.
- void setVal (coordinate val) override

  *Sets the value of the DiscreteParameter to the discrete value closest to the given value by modifying times using setTimes.*

## Private Attributes

- int times

  *Times used in the value calculation.*
- double step

  *Difference between discrete values.*
- double offset = 0

  *Offset used in the value calculation.*

### 9.14.1 Detailed Description

Implements a Parameter using discrete values.
The value of the Parameter is calculated as $val = times \cdot step + offset$.
Definition at line 12 of file DiscreteParameter.h.

### 9.14.2 Constructor & Destructor Documentation

#### 9.14.2.1 DiscreteParameter() [1/2]

```
DiscreteParameter::DiscreteParameter (
            shared_ptr< ParameterDefinition > def,
            double step,
            double value )
```

Creates a DiscreteParameter with the given ParameterDefinition, step and value.
Checks if given value is in bounds set by the ParameterDefinition. Calculates times and offset automatically.

**Parameters**

| | |
|---|---|
| *def* | ParameterDefinition of the Parameter. |
| *step* | Difference between discrete values. |
| *value* | Initial value of the Parameter. |

Definition at line 7 of file DiscreteParameter.cpp.
References Parameter::Parameter(), Parameter::getMax(), Parameter::getMin(), offset, step, and times.
Referenced by DiscreteParameter().

#### 9.14.2.2 DiscreteParameter() [2/2]

```
DiscreteParameter::DiscreteParameter (
            shared_ptr< ParameterDefinition > def,
            double step )
```

Creates a DiscreteParameter with the given ParameterDefinition and step.
Calculates times and offset automatically.

**Parameters**

| | |
|---|---|
| *def* | ParameterDefinition of the Parameter. |
| *step* | Difference between discrete values. |

Definition at line 17 of file DiscreteParameter.cpp.
References DiscreteParameter(), Parameter::getMax(), and Parameter::getMin().

## 9.14.3 Member Function Documentation

### 9.14.3.1 getOffset()

```
double DiscreteParameter::getOffset ( ) const
```

Returns the value of offset.

**Returns**

A floating point number representing the offset.

Definition at line 37 of file DiscreteParameter.cpp.
References offset.

### 9.14.3.2 getStep()

```
double DiscreteParameter::getStep ( ) const
```

Returns the value of step.

**Returns**

A floating point number representing the difference between discrete values.

Definition at line 33 of file DiscreteParameter.cpp.
References step.

### 9.14.3.3 getTimes()

```
int DiscreteParameter::getTimes ( ) const
```

Returns the value of times.

**Returns**

An integer representing the times value.

Definition at line 21 of file DiscreteParameter.cpp.
References times.

### 9.14.3.4 getVal()

```
coordinate DiscreteParameter::getVal ( ) const  [override], [virtual]
```

Returns the current value of the DiscreteParameter as calculated by the following formula: $val = times \cdot step + offset$.

**Returns**

A coordinate representing the value of the ContinuousParameter.

Implements Parameter.
Definition at line 41 of file DiscreteParameter.cpp.
References offset, step, and times.

### 9.14.3.5 setTimes()

```
void DiscreteParameter::setTimes (
            int newTimes )
```

Sets the value of times to the given value.
Checks if value is in bounds set by ParameterDefinition.

**Parameters**

| | |
|---|---|
| *newTimes* | |

Definition at line 25 of file DiscreteParameter.cpp.
References Parameter::getMax(), Parameter::getMin(), offset, step, and times.
Referenced by setVal().

#### 9.14.3.6 setVal()

```
void DiscreteParameter::setVal (
            coordinate val ) [override], [virtual]
```

Sets the value of the DiscreteParameter to the discrete value closest to the given value by modifying times using setTimes.

**Parameters**

| | |
|---|---|
| *val* | Value to set the DiscreteParameter to. |

Implements Parameter.
Definition at line 45 of file DiscreteParameter.cpp.
References offset, setTimes(), and step.

### 9.14.4 Member Data Documentation

#### 9.14.4.1 offset

```
double DiscreteParameter::offset = 0 [private]
```

Offset used in the value calculation.
Definition at line 25 of file DiscreteParameter.h.
Referenced by DiscreteParameter(), getOffset(), getVal(), setTimes(), and setVal().

#### 9.14.4.2 step

```
double DiscreteParameter::step [private]
```

Difference between discrete values.
Used in the value calculation.
Definition at line 21 of file DiscreteParameter.h.
Referenced by DiscreteParameter(), getStep(), getVal(), setTimes(), and setVal().

#### 9.14.4.3 times

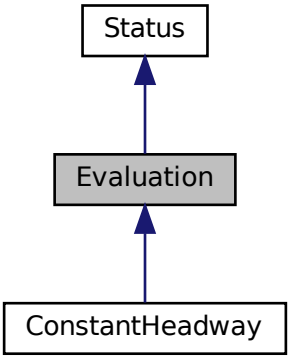
```
int DiscreteParameter::times [private]
```

Times used in the value calculation.
Definition at line 17 of file DiscreteParameter.h.
Referenced by DiscreteParameter(), getTimes(), getVal(), and setTimes().
The documentation for this class was generated from the following files:

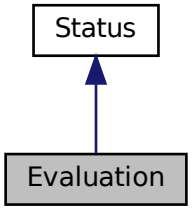
- /home/runner/work/simopticon/simopticon/src/parameters/DiscreteParameter.h
- /home/runner/work/simopticon/simopticon/src/parameters/DiscreteParameter.cpp

## 9.15 Evaluation Class Reference

A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization.
`#include "Evaluation.h"`
Inheritance diagram for Evaluation:




Collaboration diagram for Evaluation:




### Public Member Functions

- virtual functionValue processOutput (filesystem::path path, set< runId > experimentIds)=0

  *Returns a value to the results of a single simulation run.*
- virtual map< pair< filesystem::path, set< runId > >, functionValue > processOutput (const set< pair< filesystem::path, set< runId >>> &experimentResults)

  *Returns values to the results of multiple simulation runs.*
- string getName () override

  *Returns a string representing the name of the implementing component in natural language.*
- string getStatus () override

  *Returns a string representing the current state of the implementing component.*
- string getStatusBar () override

  *Returns a string representing the current progress of the calculations of the implementing component.*

## Additional Inherited Members

### 9.15.1 Detailed Description

A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization.
A lower value is considered better in this framework. The optimized function can be viewed as an error function.
Definition at line 24 of file Evaluation.h.

### 9.15.2 Member Function Documentation

#### 9.15.2.1 getName()

`string Evaluation::getName ( ) [override], [virtual]`
Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented from Status.
Definition at line 12 of file Evaluation.cpp.
References Status::getName().

#### 9.15.2.2 getStatus()

`string Evaluation::getStatus ( ) [override], [virtual]`
Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented from Status.
Definition at line 16 of file Evaluation.cpp.
References Status::getStatus().

#### 9.15.2.3 getStatusBar()

`string Evaluation::getStatusBar ( ) [override], [virtual]`
Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented from Status.
Definition at line 20 of file Evaluation.cpp.
References Status::getStatusBar().

#### 9.15.2.4 processOutput() [1/2]

```
map< pair< filesystem::path, set< runId > >, functionValue > Evaluation::processOutput (
            const set< pair< filesystem::path, set< runId >>> & experimentResults ) [virtual]
```
Returns values to the results of multiple simulation runs.
Simply calls processOutput(filesystem::path, set<runId>) multiple times if not overridden.

**Parameters**

| | |
|---|---|
| *experimentResults* | Paths to and identifiers of the simulation results. |

**Returns**

A map which maps the given results to their respective performance value.

Reimplemented in ConstantHeadway.
Definition at line 4 of file Evaluation.cpp.

#### 9.15.2.5 processOutput() [2/2]

```
virtual functionValue Evaluation::processOutput (
            filesystem::path path,
            set< runId > experimentIds )  [pure virtual]
```

Returns a value to the results of a single simulation run.

**Parameters**

| | |
|---|---|
| *path* | Path to the result files. |
| *experimentIds* | Identifiers of certain simulation runs within the directory represented by the given path. |

**Returns**

A value that represents the performance of the simulation - the lower the better.

Implemented in ConstantHeadway.
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/evaluation/Evaluation.h
- /home/runner/work/simopticon/simopticon/src/evaluation/Evaluation.cpp

## 9.16 GrahamScan Class Reference

A class providing functionality for finding the lower right convex hull of a set of points.
`#include "GrahamScan.h"`

### Static Public Member Functions

- static list< pair< shared_ptr< HyRect >, double > > scan (list< shared_ptr< HyRect >> vertices)

  *Calculates the lower right convex hull of a set of points.*

### 9.16.1 Detailed Description

A class providing functionality for finding the lower right convex hull of a set of points.
Definition at line 15 of file GrahamScan.h.

### 9.16.2 Member Function Documentation

#### 9.16.2.1 scan()

```
list< pair< shared_ptr< HyRect >, double > > GrahamScan::scan (
            list< shared_ptr< HyRect >> vertices )  [static]
```

Calculates the lower right convex hull of a set of points.
Points are defined by the given HyRects diagonal length (x axis) and average value (y axis). For each returned HyRect the slope to the point right of it is returned (if it is the rightmost point, infinity is chosen). That slope value can be used by DIRECT as the highest Lipschitz constant for which the HyRect satisfies the first condition.

**Parameters**

| | |
|---|---|
| *vertices* | List of rectangles with different sizes. |

**Returns**

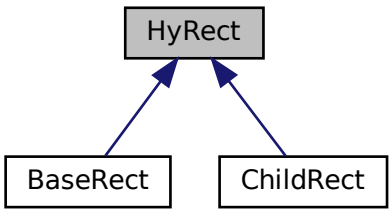
A list of rectangles and corresponding Lipschitz constants that represents convex hull meaning a subset of the given HyRect list.

Definition at line 7 of file GrahamScan.cpp.
References HyRect::getAvgValue(), HyRect::getDepth(), and HyRect::getDiagonalLength().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/GrahamScan.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/GrahamScan.cpp

## 9.17 HyRect Class Reference

An abstract class representing a rectangular part of the search space.
`#include "HyRect.h"`
Inheritance diagram for HyRect:




### Public Member Functions

- HyRect (dimension D, position pos, depth t)

  *Creates a HyRect with the given dimensionality, position and depth.*
- virtual array< vector< dirCoordinate >, 2 > getSamplingVertices ()=0

  *Returns the coordinates of two opposite corner points of the rectangle.*
- dirCoordinate getDiagonalLength () const

  *Returns the length of the diagonal of the rectangle.*
- depth getDepth () const

  *Returns the value of t.*
- position getPos () const

  *Returns the value of pos.*
- dimension getSplitDim () const

  *Calculates the dimension where this rectangle must be or has been split by divide.*

- functionValue getAvgValue () const

  *Returns the value of avgValue.*
- dimension getD () const

  *Returns the value of D.*
- void setAvgValue (functionValue value)

  *Sets the value of avgValue.*
- virtual bool operator== (const HyRect &rect) const

  *Checks if the current and the given HyRect objects are equal by comparing their pos, D, and t.*
- bool operator< (const HyRect &rect) const

  *Compares depth t and avgValue of the given HyRect objects.*
- bool operator!= (const HyRect &rhs) const

  *Checks if the current and the given HyRect objects are unequal by comparing their pos, D, and t.*
- bool operator> (const HyRect &rhs) const

  *Compares depth t and avgValue of the given HyRect objects.*
- bool operator<= (const HyRect &rhs) const

  *Compares depth t and avgValue of the given HyRect objects.*
- bool operator>= (const HyRect &rhs) const

  *Compares depth t and avgValue of the given HyRect objects.*

## Static Public Member Functions

- static array< shared_ptr< HyRect >, 3 > divide (const shared_ptr< HyRect > &ptr)

  *Divides the given rectangle into three smaller ChildRect which take the given HyRect as a parent.*

## Protected Attributes

- dimension D

  *Dimensionality of the rectangle.*
- depth t

  *Depth of the rectangle in the partition tree.*
- position pos

  *Position of the rectangle relative to its parent rectangle.*
- functionValue avgValue = INFINITY

  *Mean between the values obtained at the parameters returned by getSamplingVertices.*

### 9.17.1 Detailed Description

An abstract class representing a rectangular part of the search space.
Definition at line 35 of file HyRect.h.

### 9.17.2 Constructor & Destructor Documentation

#### 9.17.2.1 HyRect()

```
HyRect::HyRect (
            dimension D,
            position pos,
            depth t )
```

Creates a HyRect with the given dimensionality, position and depth.

**Parameters**

| | |
|---|---|
| *D* | Dimensionality of the rectangle (i.e. the search space). |

**Parameters**

| | |
|---|---|
| *pos* | Position relative to parent rectangle. |
| *t* | Depth of the rectangle in partition tree. |

Definition at line 6 of file HyRect.cpp.
References D, pos, and t.
Referenced by BaseRect::BaseRect(), and ChildRect::ChildRect().

### 9.17.3 Member Function Documentation

#### 9.17.3.1 divide()

```
array< shared_ptr< HyRect >, 3 > HyRect::divide (
            const shared_ptr< HyRect > & ptr )  [static]
```

Divides the given rectangle into three smaller ChildRect which take the given HyRect as a parent.

**Parameters**

| | |
|---|---|
| *ptr* | Reference to a shared_ptr to the HyRect that is being divided. |

**Returns**

An array of ChildRect instances generated by dividing the given HyRect.

Definition at line 9 of file HyRect.cpp.
References ChildRect::ChildRect().

#### 9.17.3.2 getAvgValue()

```
functionValue HyRect::getAvgValue ( ) const
```

Returns the value of avgValue.

**Returns**

A functionValue representing the average value on the sampled corners of the rectangle.

Definition at line 35 of file HyRect.cpp.
References avgValue.
Referenced by DirectOptimizer::estimatedValue(), CmpSharedHyrect::operator()(), and GrahamScan::scan().

#### 9.17.3.3 getD()

```
dimension HyRect::getD ( ) const
```

Returns the value of D.

**Returns**

A dimension representing the number of dimensions of the rectangle.

Definition at line 67 of file HyRect.cpp.
References D.
Referenced by ChildRect::ChildRect().

#### 9.17.3.4 getDepth()

```
depth HyRect::getDepth ( ) const
```
Returns the value of t.

**Returns**

A depth value representing the depth of the rectangle in the partition tree.

Definition at line 31 of file HyRect.cpp.
References t.
Referenced by ChildRect::ChildRect(), and GrahamScan::scan().

#### 9.17.3.5 getDiagonalLength()

```
dirCoordinate HyRect::getDiagonalLength ( ) const
```
Returns the length of the diagonal of the rectangle.
Basically calculates the euclidian distance between the vertices returned by getSamplingVertices. Instead of actually invoking the costly recursive getSamplingVertices function, a calculation based on t is executed

**Returns**

A dirCoordinate representing the diagonal length of the rectangle.

Definition at line 16 of file HyRect.cpp.
References D, and t.
Referenced by DirectOptimizer::estimatedValue(), and GrahamScan::scan().

#### 9.17.3.6 getPos()

```
position HyRect::getPos ( ) const
```
Returns the value of pos.

**Returns**

A position value representing the relative position to the parent rectangle.

Definition at line 27 of file HyRect.cpp.
References pos.
Referenced by ChildRect::operator==().

#### 9.17.3.7 getSamplingVertices()

```
virtual array<vector<dirCoordinate>, 2> HyRect::getSamplingVertices ( )  [pure virtual]
```
Returns the coordinates of two opposite corner points of the rectangle.
The returned vertices must be sampled.

**Returns**

An array containing two dirCoordinate vectors of the sampled vertices.

Implemented in ChildRect, and BaseRect.
Referenced by ChildRect::getSamplingVertices(), and CmpSharedHyrect::operator()().

#### 9.17.3.8 getSplitDim()

```
dimension HyRect::getSplitDim ( ) const
```
Calculates the dimension where this rectangle must be or has been split by divide.
Since the split dimensions are simply chosen in ascending order the calculations only needs the depth stored in t.

**Returns**

A dimension where the HyRect has been oder will be split.

Definition at line 23 of file HyRect.cpp.
References D, and t.
Referenced by ChildRect::getSamplingVertices().

#### 9.17.3.9 operator"!=()

```
bool HyRect::operator!= (
            const HyRect & rhs ) const
```

Checks if the current and the given HyRect objects are unequal by comparing their pos, D, and t.
Basically negates operator==.

**Parameters**

| | |
|---|---|
| *rhs* | HyRect to be compared. |

**Returns**

A boolean defining if the HyRect objects have different positions in the partition tree.

Definition at line 51 of file HyRect.cpp.
References operator==().

#### 9.17.3.10 operator<()

```
bool HyRect::operator< (
            const HyRect & rect ) const
```

Compares depth t and avgValue of the given HyRect objects.

**Parameters**

| | |
|---|---|
| *rect* | HyRect to be compared. |

**Returns**

A boolean defining if the depth t of this HyRect is greater than that of the given HyRect or whether the avgValue is less than that of the given HyRect if depth t is the same.

Definition at line 47 of file HyRect.cpp.
References avgValue, and t.
Referenced by operator<=(), operator>(), and operator>=().

#### 9.17.3.11 operator<=()

```
bool HyRect::operator<= (
            const HyRect & rhs ) const
```

Compares depth t and avgValue of the given HyRect objects.
Basically negates operator>.

**Parameters**

| | |
|---|---|
| *rhs* | HyRect to be compared. |

**Returns**

A boolean defining if the depth t of this HyRect is greater than or equal to that of the given HyRect or whether the avgValue is less than or equal that of the given HyRect if depth t is the same.

Definition at line 59 of file HyRect.cpp.
References operator<().

#### 9.17.3.12 operator==()

```
bool HyRect::operator== (
            const HyRect & rect ) const  [virtual]
```

Checks if the current and the given HyRect objects are equal by comparing their pos, D, and t.

**Parameters**

| | |
|---|---|
| *rect* | HyRect to be compared. |

**Returns**

A boolean defining if the HyRect objects have the same position in the partition tree.

Reimplemented in ChildRect.
Definition at line 43 of file HyRect.cpp.
References D, pos, and t.
Referenced by operator!=().

#### 9.17.3.13 operator>()

```
bool HyRect::operator> (
            const HyRect & rhs ) const
```

Compares depth t and avgValue of the given HyRect objects.
Basically calls operator< on the switched inputs.

**Parameters**

| | |
|---|---|
| *rhs* | HyRect to be compared. |

**Returns**

A boolean defining if the depth t of this HyRect is less or equal than that of the given HyRect or whether the avgValue is greater than or equal that of the given HyRect if depth t is the same.

Definition at line 55 of file HyRect.cpp.
References operator<().

#### 9.17.3.14 operator>=()

```
bool HyRect::operator>= (
            const HyRect & rhs ) const
```

Compares depth t and avgValue of the given HyRect objects.
Basically negates operator<.

**Parameters**

| | |
|---|---|
| *rhs* | HyRect to be compared. |

**Returns**

A boolean defining if the depth t of this HyRect is less than or equal that of the given HyRect or whether the avgValue is greater than or equal that of the given HyRect if depth t is the same.

Definition at line 63 of file HyRect.cpp.
References operator<().

#### 9.17.3.15 setAvgValue()

```
void HyRect::setAvgValue (
            functionValue value )
```

Sets the value of avgValue.

**Parameters**

| | |
|---|---|
| *value* | Average value sampled at the corners of the rectangle. |

Definition at line 39 of file HyRect.cpp.
References avgValue.

### 9.17.4 Member Data Documentation

#### 9.17.4.1 avgValue

```
functionValue HyRect::avgValue = INFINITY  [protected]
```

Mean between the values obtained at the parameters returned by getSamplingVertices.
Definition at line 55 of file HyRect.h.
Referenced by getAvgValue(), operator<(), and setAvgValue().

#### 9.17.4.2 D

```
dimension HyRect::D  [protected]
```

Dimensionality of the rectangle.
Is equivalent to the dimensionality of the search space, i.e. the number of optimized parameters.
Definition at line 41 of file HyRect.h.
Referenced by HyRect(), getD(), getDiagonalLength(), BaseRect::getSamplingVertices(), getSplitDim(), and operator==().

#### 9.17.4.3 pos

```
position HyRect::pos  [protected]
```

Position of the rectangle relative to its parent rectangle.
For BaseRect, *pos* is always BASE.
Definition at line 51 of file HyRect.h.
Referenced by HyRect(), getPos(), ChildRect::getSamplingVertices(), operator==(), and ChildRect::operator==().

#### 9.17.4.4 t

```
depth HyRect::t  [protected]
```

Depth of the rectangle in the partition tree.
Equal to the number of transitive parent rectangles. For BaseRect, *t* is always 0.
Definition at line 46 of file HyRect.h.
Referenced by HyRect(), getDepth(), getDiagonalLength(), getSplitDim(), operator<(), and operator==().

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

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/HyRect.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/HyRect.cpp

## 9.18 Levels Class Reference

A providing functionality for the usage of different weightings between local and global search throughout the optimization using different levels.
`#include "Levels.h"`

### Public Member Functions

- unsigned char nextLevel ()

  *Switches currentLevel to the next local level if global is false.*
- list< shared_ptr< HyRect > > getRectSubset (const map< depth, set< shared_ptr< HyRect >, CmpSharedHyrect >, greater<>> &rects, size_t size) const

  *Calculates the subset of all given rectangles based on the current level and returns a list containing only the best HyRect per diagonal length.*
- double getEpsilon () const

  *Returns the epsilon value on the current level the DIRECT algorithm resides on.*
- unsigned char getLevel () const

  *Returns a number corresponding to the current level the optimization resides on.*
- bool isGlobal () const

  *Returns the value of global.*
- void setGlobal (bool val)

  *Sets the value of global.*

### Static Public Attributes

- constexpr static const double L3_EPSILON = 1e-5

  *Epsilon value to be used when DIRECT algorithm uses level 3.*
- constexpr static const double L2_EPSILON = 1e-5

  *Epsilon value to be used when DIRECT algorithm uses level 2.*
- constexpr static const double L1_EPSILON = 1e-7

  *Epsilon value to be used when DIRECT algorithm uses level 1.*
- constexpr static const double L0_EPSILON = 0

  *Epsilon value to be used when DIRECT algorithm uses level 0.*
- constexpr static const long double L3_SIZE = 0.5

  *Fraction of rectangles in partition to be used on level 3 (only larger rectangles are considered).*
- constexpr static const long double L2_SIZE = 1

  *Fraction of rectangles in partition to be used on level 2 (only smaller rectangles are considered).*
- constexpr static const long double L1_SIZE = 0.95

  *Fraction of rectangles in partition to be used on level 1 (only smaller rectangles are considered).*
- constexpr static const long double L0_SIZE = 0.04

  *Fraction of rectangles in partition to be used on level 0 (only smaller rectangles are considered).*

### Private Attributes

- level currentLevel = l2_0

  *Local level the optimization is currently using when global is false.*
- bool global = false

  *Defines whether global optimization (level 3) or one of the local levels (0-2) is used.*

## 9.18.1 Detailed Description

A providing functionality for the usage of different weightings between local and global search throughout the optimization using different levels.
Definition at line 26 of file Levels.h.

## 9.18.2 Member Function Documentation

### 9.18.2.1 getEpsilon()

```
double Levels::getEpsilon ( ) const
```

Returns the epsilon value on the current level the DIRECT algorithm resides on.
Either L3_EPSILON, L2_EPSILON, L1_EPSILON or L0_EPSILON.

**Returns**

A floating point value used as epsilon parameter on the current level.

Definition at line 52 of file Levels.cpp.
References getLevel(), L0_EPSILON, L1_EPSILON, L2_EPSILON, and L3_EPSILON.

### 9.18.2.2 getLevel()

```
unsigned char Levels::getLevel ( ) const
```

Returns a number corresponding to the current level the optimization resides on.

**Returns**

An integral corresponding to the current level.

Definition at line 65 of file Levels.cpp.
References currentLevel, and global.
Referenced by getEpsilon(), getRectSubset(), DirectOptimizer::getStatus(), and nextLevel().

### 9.18.2.3 getRectSubset()

```
list< shared_ptr< HyRect > > Levels::getRectSubset (
            const map< depth, set< shared_ptr< HyRect >, CmpSharedHyrect >, greater<>> &
rects,
            size_t size ) const
```

Calculates the subset of all given rectangles based on the current level and returns a list containing only the best HyRect per diagonal length.

**Parameters**

| | |
|---|---|
| *rects* | Map containing all HyRect of the current partition grouped by HyRect::t and sorted by HyRect::avgValue. |
| *size* | Number of HyRect in the given partition. |

**Returns**

A list containing only the best HyRect per diagonal length in the subset based on the current level.

Definition at line 15 of file Levels.cpp.
References getLevel(), global, L0_SIZE, L1_SIZE, L2_SIZE, and L3_SIZE.

#### 9.18.2.4 isGlobal()

```
bool Levels::isGlobal ( ) const
```
Returns the value of global.

**Returns**

A boolean defining whether the optimization is currently in the global phase.

Definition at line 81 of file Levels.cpp.
References global.
Referenced by DirectOptimizer::runOptimization().

#### 9.18.2.5 nextLevel()

```
unsigned char Levels::nextLevel ( )
```
Switches currentLevel to the next local level if global is false.

**Returns**

A number representing the current level after switching.

Definition at line 7 of file Levels.cpp.
References currentLevel, getLevel(), and global.
Referenced by DirectOptimizer::runOptimization().

#### 9.18.2.6 setGlobal()

```
void Levels::setGlobal (
            bool val )
```
Sets the value of global.

**Parameters**

| | |
|---|---|
| *val* | Defines whether global optimization should be used in the following iterations. |

Definition at line 85 of file Levels.cpp.
References global.
Referenced by DirectOptimizer::runOptimization().

### 9.18.3 Member Data Documentation

#### 9.18.3.1 currentLevel

```
level Levels::currentLevel = l2_0  [private]
```
Local level the optimization is currently using when global is *false*.
Definition at line 31 of file Levels.h.
Referenced by getLevel(), and nextLevel().

#### 9.18.3.2 global

```
bool Levels::global = false  [private]
```
Defines whether global optimization (level 3) or one of the local levels (0-2) is used.
Definition at line 35 of file Levels.h.
Referenced by getLevel(), getRectSubset(), isGlobal(), nextLevel(), and setGlobal().

#### 9.18.3.3 L0_EPSILON

```
constexpr static const double Levels::L0_EPSILON = 0  [static], [constexpr]
```
Epsilon value to be used when DIRECT algorithm uses level 0.
Definition at line 53 of file Levels.h.
Referenced by getEpsilon().

#### 9.18.3.4 L0_SIZE

```
constexpr static const long double Levels::L0_SIZE = 0.04  [static], [constexpr]
```
Fraction of rectangles in partition to be used on level 0 (only smaller rectangles are considered).
Definition at line 70 of file Levels.h.
Referenced by getRectSubset().

#### 9.18.3.5 L1_EPSILON

```
constexpr static const double Levels::L1_EPSILON = 1e-7  [static], [constexpr]
```
Epsilon value to be used when DIRECT algorithm uses level 1.
Definition at line 49 of file Levels.h.
Referenced by getEpsilon().

#### 9.18.3.6 L1_SIZE

```
constexpr static const long double Levels::L1_SIZE = 0.95  [static], [constexpr]
```
Fraction of rectangles in partition to be used on level 1 (only smaller rectangles are considered).
Definition at line 66 of file Levels.h.
Referenced by getRectSubset().

#### 9.18.3.7 L2_EPSILON

```
constexpr static const double Levels::L2_EPSILON = 1e-5  [static], [constexpr]
```
Epsilon value to be used when DIRECT algorithm uses level 2.
Definition at line 45 of file Levels.h.
Referenced by getEpsilon().

#### 9.18.3.8 L2_SIZE

```
constexpr static const long double Levels::L2_SIZE = 1  [static], [constexpr]
```
Fraction of rectangles in partition to be used on level 2 (only smaller rectangles are considered).
Definition at line 62 of file Levels.h.
Referenced by getRectSubset().

#### 9.18.3.9 L3_EPSILON

```
constexpr static const double Levels::L3_EPSILON = 1e-5  [static], [constexpr]
```
Epsilon value to be used when DIRECT algorithm uses level 3.

Definition at line 41 of file Levels.h.
Referenced by getEpsilon(), and DirectOptimizer::runOptimization().

#### 9.18.3.10 L3_SIZE

```
constexpr static const long double Levels::L3_SIZE = 0.5  [static], [constexpr]
```

Fraction of rectangles in partition to be used on level 3 (only larger rectangles are considered).
Definition at line 58 of file Levels.h.
Referenced by getRectSubset().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/Levels.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/Levels.cpp

## 9.19 Multithreaded< Key, T, Compare, Allocator > Class Template Reference

A class implementing concurrent execution of the same function for different arguments.
`#include "Multithreaded.h"`

### Public Member Functions

- Multithreaded (unsigned int threads)

  *Creates a Multithreaded class that does not use more than the given number of threads.*

### Protected Member Functions

- virtual map< Key, T, Compare, Allocator > runMultithreadedFunctions (set< Key, Compare > runs)

  *Pushes given tasks into queue, creates concurrent threads and merges them when execution is done.*
- virtual map< Key, T, Compare, Allocator > multithreadFunction ()

  *Function that is executed by each thread.*

### Protected Attributes

- const unsigned int NR_THREADS

  *Maximum number of concurrent threads to be used in ThreadPool.*
- ThreadsafeQueue< Key > queue

  *ThreadsafeQueue containing the arguments that have to be processed by the ThreadPool.*

### Private Member Functions

- virtual T work (Key arg)=0

  *Function that should be executed concurrently on different arguments.*

### 9.19.1 Detailed Description

**template<class Key, class T, class Compare = less<Key>, class Allocator = allocator<pair<const Key, T>>>**
**class Multithreaded< Key, T, Compare, Allocator >**

A class implementing concurrent execution of the same function for different arguments.
The function must be implemented through work and execution follows the ThreadPool design pattern.

**Template Parameters**

| | |
|---|---|
| *Key* | Argument type of the concurrent work function. |

**Template Parameters**

| | |
|---|---|
| *T* | Result type of the concurrent work function. |
| *Compare* | Comparison for objects of type Key. |
| *Allocator* | Allocator for pairs of constant Key and T. |

Definition at line 24 of file Multithreaded.h.

## 9.19.2 Constructor & Destructor Documentation

### 9.19.2.1 Multithreaded()

```
template<class Key , class T , class Compare = less<Key>, class Allocator = allocator<pair<const
Key, T>>>
Multithreaded< Key, T, Compare, Allocator >::Multithreaded (
            unsigned int threads )  [explicit]
```

Creates a Multithreaded class that does not use more than the given number of threads.

**Parameters**

| | |
|---|---|
| *threads* | Maximum number of threads to use. |

## 9.19.3 Member Function Documentation

### 9.19.3.1 multithreadFunction()

```
template<class Key , class T , class Compare = less<Key>, class Allocator = allocator<pair<const
Key, T>>>
virtual map<Key, T, Compare, Allocator> Multithreaded< Key, T, Compare, Allocator >::multithread↩
Function ( )  [protected], [virtual]
```

Function that is executed by each thread.
As long as queue is not empty, tasks are started. When queue is empty, the processed results are returned

**Returns**

A map which maps arguments to their respective calculated values.

### 9.19.3.2 runMultithreadedFunctions()

```
template<class Key , class T , class Compare = less<Key>, class Allocator = allocator<pair<const
Key, T>>>
virtual map<Key, T, Compare, Allocator> Multithreaded< Key, T, Compare, Allocator >::run↩
MultithreadedFunctions (
            set< Key, Compare > runs )  [protected], [virtual]
```

Pushes given tasks into queue, creates concurrent threads and merges them when execution is done.

**Parameters**

| | |
|---|---|
| *runs* | Set of arguments on which work should to be executed. |

**Returns**

A map which maps arguments to their respective calculated values.

#### 9.19.3.3 work()

```
template<class Key , class T , class Compare = less<Key>, class Allocator = allocator<pair<const
Key, T>>>
virtual T Multithreaded< Key, T, Compare, Allocator >::work (
            Key arg )  [private], [pure virtual]
```

Function that should be executed concurrently on different arguments.

**Parameters**

| | |
|---|---|
| *arg* | Argument of the concurrently executed function. |

**Returns**

Return value of the concurrently executed function.

### 9.19.4 Member Data Documentation

#### 9.19.4.1 NR_THREADS

```
template<class Key , class T , class Compare = less<Key>, class Allocator = allocator<pair<const
Key, T>>>
const unsigned int Multithreaded< Key, T, Compare, Allocator >::NR_THREADS  [protected]
```

Maximum number of concurrent threads to be used in ThreadPool.
Definition at line 37 of file Multithreaded.h.

#### 9.19.4.2 queue

```
template<class Key , class T , class Compare = less<Key>, class Allocator = allocator<pair<const
Key, T>>>
ThreadsafeQueue<Key> Multithreaded< Key, T, Compare, Allocator >::queue  [protected]
```

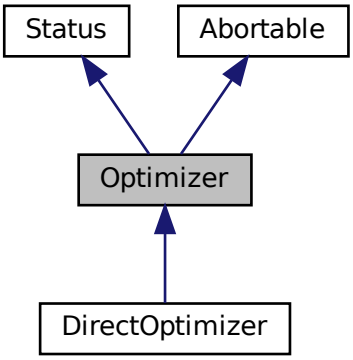
ThreadsafeQueue containing the arguments that have to be processed by the ThreadPool.
Definition at line 41 of file Multithreaded.h.
The documentation for this class was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/utils/Multithreaded.h

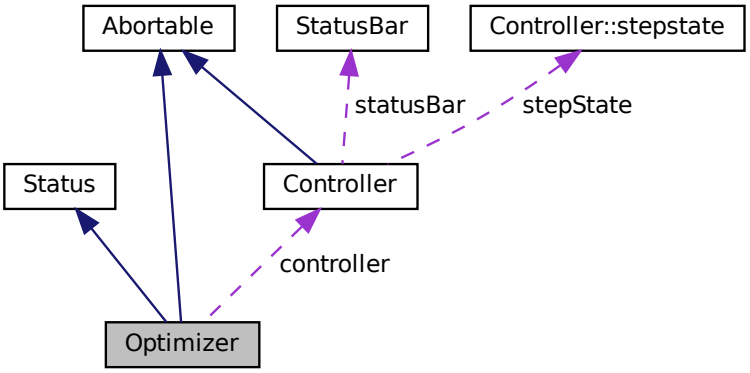
## 9.20 Optimizer Class Reference

A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.
`#include "Optimizer.h"`

Inheritance diagram for Optimizer:




Collaboration diagram for Optimizer:




## Public Member Functions

- Optimizer (Controller &ctrl, list< shared_ptr< ParameterDefinition >> params)

  *Creates an Optimizer which can request values from the given Controller and tries to optimize the given parameters.*
- virtual void runOptimization ()=0

  *Starts the optimization process.*
- ValueMap & getValueMap () const

  *Returns a reference to Controller::valueMap.*
- string getName () override

  *Returns a string representing the name of the implementing component in natural language.*
- string getStatus () override

  *Returns a string representing the current state of the implementing component.*
- string getStatusBar () override

  *Returns a string representing the current progress of the calculations of the implementing component.*

## Protected Member Functions

- map< vector< shared_ptr< Parameter > >, functionValue > requestValues (const list< vector< shared_ptr< Parameter >>> &params)

  *Requests the values when using certain Parameter combinations from controller.*

## Protected Attributes

- list< shared_ptr< ParameterDefinition > > parameters

  *List of parameters to be optimized.*

## Private Attributes

- Controller & controller

  *Reference to the executing Controller to be able to request values using Controller::requestValues.*

## Additional Inherited Members

### 9.20.1 Detailed Description

A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.
The Optimizer has control over which Parameter combinations are simulated and evaluated as well as the duration of the optimization. If abort is called the optimization strategy should finish the optimization as soon as possible.
Definition at line 34 of file Optimizer.h.

### 9.20.2 Constructor & Destructor Documentation

#### 9.20.2.1 Optimizer()

```
Optimizer::Optimizer (
            Controller & ctrl,
            list< shared_ptr< ParameterDefinition >> params )
```

Creates an Optimizer which can request values from the given Controller and tries to optimize the given parameters.

**Parameters**

| | |
|---|---|
| *ctrl* | Controller to be used for evaluation of Parameter combinations. |
| *params* | List of ParameterDefinition defining the parameters that must be optimized. |

Definition at line 13 of file Optimizer.cpp.
References controller, and parameters.

### 9.20.3 Member Function Documentation

#### 9.20.3.1 getName()

```
string Optimizer::getName ( )  [override], [virtual]
```

Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented from Status.
Definition at line 21 of file Optimizer.cpp.
References Status::getName().

#### 9.20.3.2 getStatus()

```
string Optimizer::getStatus ( )  [override], [virtual]
```

Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented from Status.
Definition at line 25 of file Optimizer.cpp.
References Status::getStatus().

#### 9.20.3.3 getStatusBar()

```
string Optimizer::getStatusBar ( )  [override], [virtual]
```

Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented from Status.
Definition at line 29 of file Optimizer.cpp.
References Status::getStatusBar().

#### 9.20.3.4 getValueMap()

```
ValueMap & Optimizer::getValueMap ( ) const
```

Returns a reference to Controller::valueMap.
Basically calls Controller::getValueMap on controller.

**Returns**

Definition at line 17 of file Optimizer.cpp.
References controller, and Controller::getValueMap().
Referenced by DirectOptimizer::runOptimization().

#### 9.20.3.5 requestValues()

```
map< vector< shared_ptr< Parameter > >, functionValue > Optimizer::requestValues (
            const list< vector< shared_ptr< Parameter >>> & params )  [protected]
```

Requests the values when using certain Parameter combinations from controller.
Basically calls Controller::requestValues with the given values.

**Parameters**

| | |
|---|---|
| *params* | Parameter combinations to be evaluated. |

**Returns**

A map which maps Parameter combinations to their respective values.

Definition at line 9 of file Optimizer.cpp.
References controller, and Controller::requestValues().

#### 9.20.3.6 runOptimization()

```
virtual void Optimizer::runOptimization ( )  [pure virtual]
```

Starts the optimization process.
Should only return if the optimization strategy deems the optimization complete or when abort is called.
Implemented in DirectOptimizer.
Referenced by Controller::run().

### 9.20.4 Member Data Documentation

#### 9.20.4.1 controller

```
Controller& Optimizer::controller  [private]
```

Reference to the executing Controller to be able to request values using Controller::requestValues.
Definition at line 39 of file Optimizer.h.
Referenced by Optimizer(), getValueMap(), and requestValues().

#### 9.20.4.2 parameters

```
list<shared_ptr<ParameterDefinition> > Optimizer::parameters  [protected]
```

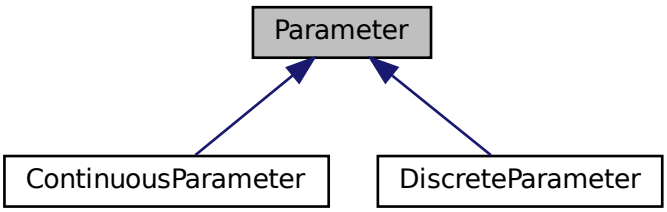
List of parameters to be optimized.
Definition at line 45 of file Optimizer.h.
Referenced by Optimizer().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/optimizer/Optimizer.h
- /home/runner/work/simopticon/simopticon/src/optimizer/Optimizer.cpp

## 9.21 Parameter Class Reference

A class acting as the container of the value of a parameter defined by a ParameterDefinition.
`#include "Parameter.h"`

Inheritance diagram for Parameter:




## Public Member Functions

- Parameter (shared_ptr< ParameterDefinition > def)

  *Creates a Parameter with the given ParameterDefinition.*
- coordinate getMin () const

  *Returns the minimum value of the Parameter stored in ParameterDefinition::min of definition.*
- coordinate getMax () const

  *Returns the maximum value of the Parameter stored in ParameterDefinition::max of definition.*
- const string & getUnit () const

  *Returns the unit string of the Parameter stored in ParameterDefinition::unit of definition.*
- const string & getConfig () const

  *Returns the configuration string of the Parameter stored in ParameterDefinition::config of definition.*
- virtual coordinate getVal () const =0

  *Returns the current value of the Parameter.*
- virtual void setVal (coordinate val)=0

  *Sets the value of the Parameter to the given value.*
- bool operator== (const Parameter &rhs) const

  *Checks if the current and the given Parameter objects are equal by comparing their value and definition.*
- bool operator!= (const Parameter &rhs) const

  *Checks if the current and the given Parameter objects are unequal by comparing their value and definition.*
- bool operator< (const Parameter &rhs) const

  *Compares the value of the given Parameter objects.*
- bool operator> (const Parameter &rhs) const

  *Compares the value of the given Parameter objects.*
- bool operator<= (const Parameter &rhs) const

  *Compares the value of the given Parameter objects.*
- bool operator>= (const Parameter &rhs) const

  *Compares the value of the given Parameter objects.*

## Private Attributes

- shared_ptr< ParameterDefinition > definition

  *Reference to the defining ParameterDefinition.*

### 9.21.1 Detailed Description

A class acting as the container of the value of a parameter defined by a ParameterDefinition.
Definition at line 23 of file Parameter.h.

## 9.21.2 Constructor & Destructor Documentation

### 9.21.2.1 Parameter()

```
Parameter::Parameter (
            shared_ptr< ParameterDefinition > def )  [explicit]
```

Creates a Parameter with the given ParameterDefinition.

**Parameters**

| | |
|---|---|
| *def* | Definition of properties of the Parameter. |

Definition at line 7 of file Parameter.cpp.
References definition.
Referenced by ContinuousParameter::ContinuousParameter(), and DiscreteParameter::DiscreteParameter().

## 9.21.3 Member Function Documentation

### 9.21.3.1 getConfig()

```
const string & Parameter::getConfig ( ) const
```

Returns the configuration string of the Parameter stored in ParameterDefinition::config of definition.

**Returns**

A string reference containing the configuration.

Definition at line 23 of file Parameter.cpp.
References definition, and ParameterDefinition::getConfig().
Referenced by StatusBar::printResult().

### 9.21.3.2 getMax()

```
coordinate Parameter::getMax ( ) const
```

Returns the maximum value of the Parameter stored in ParameterDefinition::max of definition.

**Returns**

A coordinate representing the maximum value.

Definition at line 15 of file Parameter.cpp.
References definition, and ParameterDefinition::getMax().
Referenced by ContinuousParameter::ContinuousParameter(), DiscreteParameter::DiscreteParameter(), Parameter↩
Normalizer::normalize(), DiscreteParameter::setTimes(), and ContinuousParameter::setVal().

### 9.21.3.3 getMin()

```
coordinate Parameter::getMin ( ) const
```

Returns the minimum value of the Parameter stored in ParameterDefinition::min of definition.

**Returns**

A coordinate representing the minimum value.

Definition at line 11 of file Parameter.cpp.
References definition, and ParameterDefinition::getMin().
Referenced by ContinuousParameter::ContinuousParameter(), DiscreteParameter::DiscreteParameter(), Parameter↩
Normalizer::normalize(), DiscreteParameter::setTimes(), and ContinuousParameter::setVal().

#### 9.21.3.4 getUnit()

```
const string & Parameter::getUnit ( ) const
```

Returns the unit string of the Parameter stored in ParameterDefinition::unit of definition.

**Returns**

A string reference containing the unit.

Definition at line 19 of file Parameter.cpp.
References definition, and ParameterDefinition::getUnit().
Referenced by StatusBar::printResult().

#### 9.21.3.5 getVal()

```
virtual coordinate Parameter::getVal ( ) const  [pure virtual]
```

Returns the current value of the Parameter.

**Returns**

A coordinate representing the value of the Parameter.

Implemented in DiscreteParameter, and ContinuousParameter.
Referenced by ValueMap::isTopValue(), ParameterNormalizer::normalize(), operator<(), operator==(), and StatusBar::printResult().

#### 9.21.3.6 operator"!=()

```
bool Parameter::operator!= (
            const Parameter & rhs ) const
```

Checks if the current and the given Parameter objects are unequal by comparing their value and definition.
Basically negates operator==.

**Parameters**

| | |
|---|---|
| *rhs* | Parameter to be compared. |

**Returns**

A boolean defining if the Parameter objects contain another value or another definition.

Definition at line 31 of file Parameter.cpp.
References operator==().
Referenced by CmpVectorSharedParameter::operator()().

#### 9.21.3.7 operator<()

```
bool Parameter::operator< (
            const Parameter & rhs ) const
```

Compares the value of the given Parameter objects.

**Parameters**

| | |
|---|---|
| *rhs* | Parameter to be compared. |

**Returns**

A boolean defining if the value of this Parameter is less than that of the given Parameter.

Definition at line 35 of file Parameter.cpp.
References getVal().
Referenced by CmpVectorSharedParameter::operator()(), operator<=(), operator>(), and operator>=().

#### 9.21.3.8 operator<=()

```
bool Parameter::operator<= (
            const Parameter & rhs ) const
```

Compares the value of the given Parameter objects.
Basically negates operator<.

**Parameters**

| | |
|---|---|
| *rhs* | Parameter to be compared. |

**Returns**

A boolean defining if the value of this Parameter is less than or equal to that of the given Parameter.

Definition at line 43 of file Parameter.cpp.
References operator<().

#### 9.21.3.9 operator==()

```
bool Parameter::operator== (
            const Parameter & rhs ) const
```

Checks if the current and the given Parameter objects are equal by comparing their value and definition.

**Parameters**

| | |
|---|---|
| *rhs* | Parameter to be compared. |

**Returns**

A boolean defining if the Parameter objects contain the same value for the same definition.

Definition at line 27 of file Parameter.cpp.
References definition, and getVal().
Referenced by operator!=().

#### 9.21.3.10 operator>()

```
bool Parameter::operator> (
            const Parameter & rhs ) const
```

Compares the value of the given Parameter objects.
Basically calls operator< on the switched inputs.

**Parameters**

| | |
|---|---|
| *rhs* | Parameter to be compared. |

**Returns**

A boolean defining if the value of this Parameter is greater than that of the given Parameter.

Definition at line 39 of file Parameter.cpp.
References operator<().

#### 9.21.3.11 operator>=()

```
bool Parameter::operator>= (
            const Parameter & rhs ) const
```

Compares the value of the given Parameter objects.
Basically negates operator>.

**Parameters**

| | |
|---|---|
| *rhs* | Parameter to be compared. |

**Returns**

A boolean defining if the value of this Parameter is greater than or equal to that of the given Parameter.

Definition at line 47 of file Parameter.cpp.
References operator<().

#### 9.21.3.12 setVal()

```
virtual void Parameter::setVal (
            coordinate val ) [pure virtual]
```

Sets the value of the Parameter to the given value.

**Parameters**

| | |
|---|---|
| *val* | Value to set the Parameter to. |

Implemented in DiscreteParameter, and ContinuousParameter.

### 9.21.4 Member Data Documentation

#### 9.21.4.1 definition

```
shared_ptr<ParameterDefinition> Parameter::definition [private]
```

Reference to the defining ParameterDefinition.
Definition at line 28 of file Parameter.h.
Referenced by Parameter(), getConfig(), getMax(), getMin(), getUnit(), and operator==().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/parameters/Parameter.h
- /home/runner/work/simopticon/simopticon/src/parameters/Parameter.cpp

## 9.22 ParameterDefinition Class Reference

A class storing information on the properties of parameters that are being optimized.
`#include "ParameterDefinition.h"`

## Public Member Functions

- ParameterDefinition (coordinate min, coordinate max, string config="", string unit="")

  *Creates a ParameterDefinition with the given minimum, maximum, configuration string and unit.*
- coordinate getMin () const

  *Returns the minimum value of the Parameter stored in min.*
- coordinate getMax () const

  *Returns the maximum value of the Parameter stored in max.*
- const string & getUnit () const

  *Returns the unit string of the Parameter stored in unit.*
- const string & getConfig () const

  *Returns the configuration string of the Parameter stored in config.*

## Private Attributes

- const coordinate min

  *Minimum value of the Parameter.*
- const coordinate max

  *Maximum value of the Parameter.*
- const string unit

  *Unit of the Parameter (optional).*
- const string config

  *String containing configuration details of the Parameter (optional).*

### 9.22.1 Detailed Description

A class storing information on the properties of parameters that are being optimized.
Definition at line 15 of file ParameterDefinition.h.

### 9.22.2 Constructor & Destructor Documentation

#### 9.22.2.1 ParameterDefinition()

```
ParameterDefinition::ParameterDefinition (
            coordinate min,
            coordinate max,
            string config = "",
            string unit = "" )
```

Creates a ParameterDefinition with the given minimum, maximum, configuration string and unit.

**Parameters**

| | |
|---|---|
| *min* | Minimum value of the Parameter. |
| *max* | Maximum value of the Parameter. |
| *config* | Configuration string for the Parameter (optional). |
| *unit* | Unit of the Parameter (optional) |

Definition at line 6 of file ParameterDefinition.cpp.
References config, max, min, and unit.

### 9.22.3 Member Function Documentation

#### 9.22.3.1 getConfig()

```
const string & ParameterDefinition::getConfig ( ) const
```

Returns the configuration string of the Parameter stored in config.

**Returns**

A string reference containing the configuration.

Definition at line 29 of file ParameterDefinition.cpp.
References config.
Referenced by Parameter::getConfig().

#### 9.22.3.2 getMax()

```
coordinate ParameterDefinition::getMax ( ) const
```

Returns the maximum value of the Parameter stored in max.

**Returns**

A coordinate representing the maximum value.

Definition at line 21 of file ParameterDefinition.cpp.
References max.
Referenced by ParameterNormalizer::denormalize(), and Parameter::getMax().

#### 9.22.3.3 getMin()

```
coordinate ParameterDefinition::getMin ( ) const
```

Returns the minimum value of the Parameter stored in min.

**Returns**

A coordinate representing the minimum value.

Definition at line 17 of file ParameterDefinition.cpp.
References min.
Referenced by ParameterNormalizer::denormalize(), and Parameter::getMin().

#### 9.22.3.4 getUnit()

```
const string & ParameterDefinition::getUnit ( ) const
```

Returns the unit string of the Parameter stored in unit.

**Returns**

A string reference containing the unit.

Definition at line 25 of file ParameterDefinition.cpp.
References unit.
Referenced by Parameter::getUnit().

### 9.22.4 Member Data Documentation

#### 9.22.4.1 config

```
const string ParameterDefinition::config  [private]
```

String containing configuration details of the Parameter (optional).
May be used to transfer configuration information for SimulationRunner.
Definition at line 33 of file ParameterDefinition.h.
Referenced by ParameterDefinition(), and getConfig().

#### 9.22.4.2 max

```
const coordinate ParameterDefinition::max  [private]
```
Maximum value of the Parameter.
Definition at line 24 of file ParameterDefinition.h.
Referenced by ParameterDefinition(), and getMax().

#### 9.22.4.3 min

```
const coordinate ParameterDefinition::min  [private]
```
Minimum value of the Parameter.
Definition at line 20 of file ParameterDefinition.h.
Referenced by ParameterDefinition(), and getMin().

#### 9.22.4.4 unit

```
const string ParameterDefinition::unit  [private]
```
Unit of the Parameter (optional).
Definition at line 28 of file ParameterDefinition.h.
Referenced by ParameterDefinition(), and getUnit().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/parameters/ParameterDefinition.h
- /home/runner/work/simopticon/simopticon/src/parameters/ParameterDefinition.cpp

## 9.23 ParameterNormalizer Class Reference

A class used for transforming parameters between the actual Parameter space and the unit hypercube used in DIRECT algorithm.
`#include "ParameterNormalizer.h"`

### Public Member Functions

- ParameterNormalizer (list< shared_ptr< ParameterDefinition >> parameters)

  *Creates a ParameterNormalizer with the given optimized parameters.*
- vector< shared_ptr< Parameter > > denormalize (vector< dirCoordinate > cords)

  *Transforms the given point in the unit hypercube into a Parameter combination.*

### Static Public Member Functions

- static vector< dirCoordinate > normalize (const vector< shared_ptr< Parameter >> &params)

  *Transforms the given Parameter combination into a point in the unit hypercube.*

### Private Attributes

- list< shared_ptr< ParameterDefinition > > parameters

  *ParameterDefinition of the optimized parameters.*

### 9.23.1 Detailed Description

A class used for transforming parameters between the actual Parameter space and the unit hypercube used in DIRECT algorithm.
Definition at line 22 of file ParameterNormalizer.h.

## 9.23.2 Constructor & Destructor Documentation

### 9.23.2.1 ParameterNormalizer()

```
ParameterNormalizer::ParameterNormalizer (
            list< shared_ptr< ParameterDefinition >> parameters )  [explicit]
```

Creates a ParameterNormalizer with the given optimized parameters.

**Parameters**

| | |
|---|---|
| *parameters* | ParameterDefinition of the optimized parameters. |

Definition at line 8 of file ParameterNormalizer.cpp.
References parameters.

## 9.23.3 Member Function Documentation

### 9.23.3.1 denormalize()

```
vector< shared_ptr< Parameter > > ParameterNormalizer::denormalize (
            vector< dirCoordinate > cords )
```

Transforms the given point in the unit hypercube into a Parameter combination.

**Parameters**

| | |
|---|---|
| *cords* | Point in the unit hypercube to be transformed. |

**Returns**

A Parameter combination corresponding to the given point in the unit hypercube.

Definition at line 20 of file ParameterNormalizer.cpp.
References ContinuousParameter::ContinuousParameter(), ParameterDefinition::getMax(), ParameterDefinition::getMin(), and parameters.

### 9.23.3.2 normalize()

```
vector< dirCoordinate > ParameterNormalizer::normalize (
            const vector< shared_ptr< Parameter >> & params )  [static]
```

Transforms the given Parameter combination into a point in the unit hypercube.

**Parameters**

| | |
|---|---|
| *params* | Parameter combination to be transformed. |

**Returns**

A point in the unit hypercube corresponding to the given Parameter combination.

Definition at line 12 of file ParameterNormalizer.cpp.
References Parameter::getMax(), Parameter::getMin(), and Parameter::getVal().

## 9.23.4 Member Data Documentation

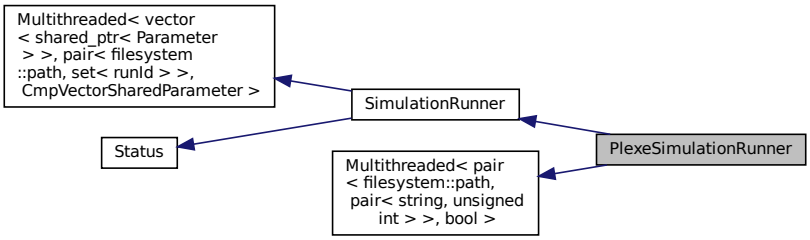
#### 9.23.4.1 parameters

`list<shared_ptr<ParameterDefinition> > ParameterNormalizer::parameters [private]`
ParameterDefinition of the optimized parameters.
Definition at line 27 of file ParameterNormalizer.h.
Referenced by ParameterNormalizer(), and denormalize().
The documentation for this class was generated from the following files:

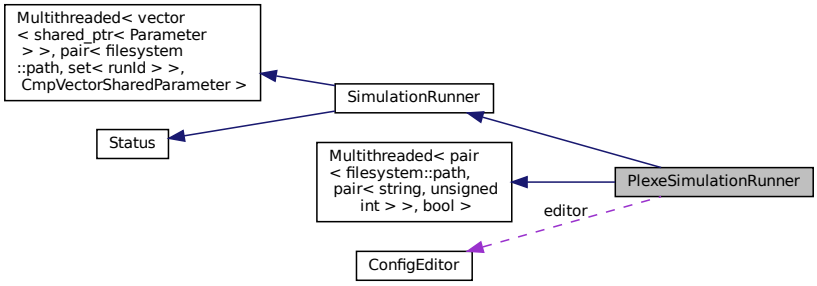
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/ParameterNormalizer.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/ParameterNormalizer.cpp

## 9.24 PlexeSimulationRunner Class Reference

A class capable of starting platooning simulations in the `Plexe` framework with given Parameter combinations.
`#include "PlexeSimulationRunner.h"`
Inheritance diagram for PlexeSimulationRunner:




Collaboration diagram for PlexeSimulationRunner:




### Public Member Functions

- PlexeSimulationRunner (unsigned int threads, unsigned int repeat, vector< string > scenarios, ConfigEditor editor)

  *Creates PlexeSimulationRunner which cannot use more than the given number of threads.*
- string getName () override

  *Returns a string representing the name of the implementing component in natural language.*
- string getStatus () override

  *Returns a string representing the current state of the implementing component.*
- string getStatusBar () override

  *Returns a string representing the current progress of the calculations of the implementing component.*

## Private Member Functions

- size_t getRunId ()

  *Returns an unique number which can be used to identify the results of a certain Parameter combination.*
- pair< filesystem::path, set< runId > > work (vector< shared_ptr< Parameter >> run) override

  *Runs simulations for the given Parameter combination.*
- bool work (std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned int >> arg) override

  *Executes one run of a Parameter combination (meaning repetition k of scenario c).*

## Private Attributes

- const unsigned int REPEAT

  *Number of repetitions per Parameter combination and scenario in SCENARIOS.*
- const vector< string > SCENARIOS

  *Scenarios that are simulated per Parameter combination.*
- ConfigEditor editor

  *ConfigEditor used for automatically creating `.ini` files with given Parameter settings.*
- size_t runNumber = 0

  *Identifier for each simulated Parameter combination.*
- mutex runNumberLock

  *Threadlock to prevent race conditions on concurrent access of runNumber.*

## Additional Inherited Members

### 9.24.1 Detailed Description

A class capable of starting platooning simulations in the `Plexe` framework with given Parameter combinations.
Definition at line 21 of file PlexeSimulationRunner.h.

### 9.24.2 Constructor & Destructor Documentation

#### 9.24.2.1 PlexeSimulationRunner()

```
PlexeSimulationRunner::PlexeSimulationRunner (
            unsigned int threads,
            unsigned int repeat,
            vector< string > scenarios,
            ConfigEditor editor )
```

Creates PlexeSimulationRunner which cannot use more than the given number of threads.
Number of repetitions, scenarios to be simulated and the ConfigEditor must also be defined. The new PlexeSimulationRunner uses $t = \min(threads, repeat \cdot size(scenarios))$ concurrent threads for parallelization of work(std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned int >>). For the parallelization of work(vector<shared_ptr<Parameter>>) $t' = \lfloor threads \div t \rfloor$ concurrent threads are used.

**Parameters**

| | |
|---|---|
| *threads* | Maximum number of threads to be used. |
| *repeat* | Number of repetitions per Parameter combination and scenario. |
| *scenarios* | Scenarios to be simulated per Parameter combination. |
| *editor* | ConfigEditor to be used. |

Definition at line 9 of file PlexeSimulationRunner.cpp.
References PlexeSimulationRunner().

Referenced by PlexeSimulationRunner().

## 9.24.3 Member Function Documentation

### 9.24.3.1 getName()

`string PlexeSimulationRunner::getName ( ) [override], [virtual]`
Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented from Status.
Definition at line 76 of file PlexeSimulationRunner.cpp.

### 9.24.3.2 getRunId()

`size_t PlexeSimulationRunner::getRunId ( ) [private]`
Returns an unique number which can be used to identify the results of a certain Parameter combination.
Returned value is only unique for one optimization process. Basically increments runNumber and returns value before incrementation.

**Returns**

An unique number used for discerning results of different runs.

Definition at line 53 of file PlexeSimulationRunner.cpp.
References runNumber, and runNumberLock.

### 9.24.3.3 getStatus()

`string PlexeSimulationRunner::getStatus ( ) [override], [virtual]`
Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented from Status.
Definition at line 80 of file PlexeSimulationRunner.cpp.
References REPEAT, and runNumber.

### 9.24.3.4 getStatusBar()

`string PlexeSimulationRunner::getStatusBar ( ) [override], [virtual]`
Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented from Status.
Definition at line 93 of file PlexeSimulationRunner.cpp.

#### 9.24.3.5 work() [1/2]

```
bool PlexeSimulationRunner::work (
            std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned
int >> arg )  [override], [private]
```

Executes one run of a Parameter combination (meaning repetition *k* of scenario *c*).
Runs command for starting Plexe and returns after execution is done.

**Parameters**

| | |
|---|---|
| *arg* | A triple containing the path to the `.ini` defining the parameters, the scenario name and the repetition number. |

**Returns**

A boolean defining whether the execution ran without throwing exceptions.

#### 9.24.3.6 work() [2/2]

```
pair< filesystem::path, set< runId > > PlexeSimulationRunner::work (
            vector< shared_ptr< Parameter >> run )  [override], [private], [virtual]
```

Runs simulations for the given Parameter combination.
Creates a new `.ini` file for the Parameter combination. Parallelizes the execution of different scenarios (see SCENARIOS) and their repetitions (see REPEAT) using Multithreaded class. Parallelized function is defined in work(std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned int >>).

**Parameters**

| | |
|---|---|
| *run* | Parameter combination to be simulated. |

**Returns**

A pair containing the path to the result files and OMNeT++-Run-IDs of the executed simulations.

Implements SimulationRunner.
Definition at line 19 of file PlexeSimulationRunner.cpp.

### 9.24.4 Member Data Documentation

#### 9.24.4.1 editor

```
ConfigEditor PlexeSimulationRunner::editor  [private]
```

ConfigEditor used for automatically creating `.ini` files with given Parameter settings.
Definition at line 38 of file PlexeSimulationRunner.h.

#### 9.24.4.2 REPEAT

```
const unsigned int PlexeSimulationRunner::REPEAT  [private]
```

Number of repetitions per Parameter combination and scenario in SCENARIOS.
Translates to repeat setting in `omnetpp.ini`. Can be set in configuration.
Definition at line 28 of file PlexeSimulationRunner.h.
Referenced by getStatus().

#### 9.24.4.3 runNumber

```
size_t PlexeSimulationRunner::runNumber = 0  [private]
```
Identifier for each simulated Parameter combination.
Is incremented when new Parameter combination is simulated. Used for unique directory names for result files.
Definition at line 44 of file PlexeSimulationRunner.h.
Referenced by getRunId(), and getStatus().

#### 9.24.4.4 runNumberLock

```
mutex PlexeSimulationRunner::runNumberLock  [private]
```
Threadlock to prevent race conditions on concurrent access of runNumber.
Definition at line 48 of file PlexeSimulationRunner.h.
Referenced by getRunId().

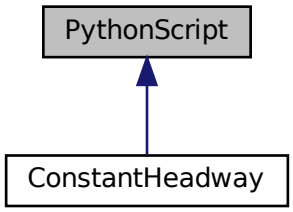
#### 9.24.4.5 SCENARIOS

```
const vector<string> PlexeSimulationRunner::SCENARIOS  [private]
```
Scenarios that are simulated per Parameter combination.
Should not invoke a GUI (e.g. pick BrakingNoGui instead of Braking). Can be set in configuration.
Definition at line 33 of file PlexeSimulationRunner.h.
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/runner/plexe/PlexeSimulationRunner.h
- /home/runner/work/simopticon/simopticon/src/runner/plexe/PlexeSimulationRunner.cpp

## 9.25 PythonScript Class Reference

A class containing functionality for interfacing with the function of a Python module on creation.
`#include "PythonScript.h"`
Inheritance diagram for PythonScript:




### Public Member Functions

- PythonScript (const std::filesystem::path &path, const char ∗functionName)

  *Creates a connection to the given function of a Python script at the given path.*
- ∼PythonScript ()

  *Ends connection to function pFunc and module pModule.*

## Protected Attributes

- PyObject * pModule

  *Pointer to module that contains function which should be used by the class.*
- PyObject * pFunc

  *Pointer to function which should be used by the class.*

### 9.25.1 Detailed Description

A class containing functionality for interfacing with the function of a Python module on creation.
See https://docs.python.org/3/c-api/index.html for more information.
Definition at line 18 of file PythonScript.h.

### 9.25.2 Constructor & Destructor Documentation

#### 9.25.2.1 PythonScript()

```
PythonScript::PythonScript (
            const std::filesystem::path & path,
            const char * functionName )
```

Creates a connection to the given function of a Python script at the given path.

**Parameters**

| | |
|---|---|
| *path* | Path to the Python script containing the function. |
| *functionName* | Name of the function to be used. |

Definition at line 8 of file PythonScript.cpp.

#### 9.25.2.2 ~PythonScript()

```
PythonScript::~PythonScript ( )
```

Ends connection to function pFunc and module pModule.
Definition at line 32 of file PythonScript.cpp.

### 9.25.3 Member Data Documentation

#### 9.25.3.1 pFunc

```
PyObject* PythonScript::pFunc  [protected]
```

Pointer to function which should be used by the class.
Definition at line 27 of file PythonScript.h.

#### 9.25.3.2 pModule

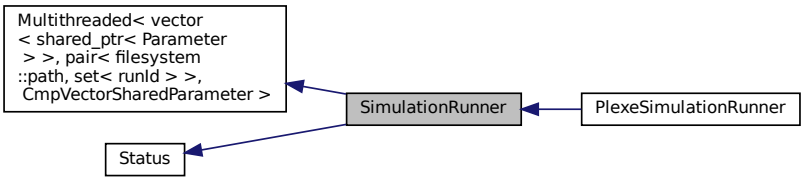
```
PyObject* PythonScript::pModule  [protected]
```

Pointer to module that contains function which should be used by the class.
Definition at line 23 of file PythonScript.h.
The documentation for this class was generated from the following files:

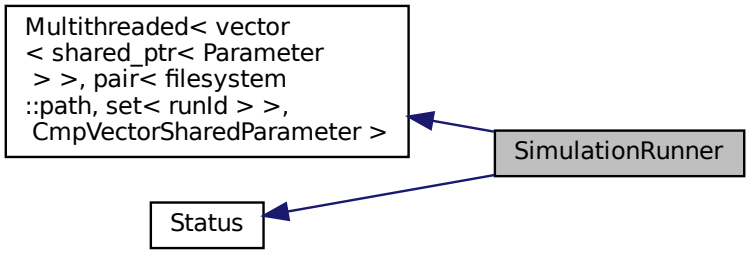
- /home/runner/work/simopticon/simopticon/src/utils/PythonScript.h
- /home/runner/work/simopticon/simopticon/src/utils/PythonScript.cpp

# 9.26 SimulationRunner Class Reference

A class capable of running simulations with certain Parameter combinations.
`#include "SimulationRunner.h"`
Inheritance diagram for SimulationRunner:




Collaboration diagram for SimulationRunner:




## Public Member Functions

- SimulationRunner (unsigned int threads)

  *Creates a SimulationRunner which can use no more than the given number of threads to simulate Parameter combinations concurrently.*
- virtual map< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, CmpVectorSharedParameter > runSimulations (const set< vector< shared_ptr< Parameter >>, CmpVectorSharedParameter > &runs)

  *Simulates the given Parameter combinations concurrently and returns their respective results.*
- string getName () override

  *Returns a string representing the name of the implementing component in natural language.*
- string getStatus () override

  *Returns a string representing the current state of the implementing component.*
- string getStatusBar () override

  *Returns a string representing the current progress of the calculations of the implementing component.*

## Private Member Functions

- pair< filesystem::path, set< runId > > work (vector< shared_ptr< Parameter >> run) override=0

  *Deals with the simulation of a single Parameter combination.*

## Additional Inherited Members

### 9.26.1 Detailed Description

A class capable of running simulations with certain Parameter combinations.
Definition at line 30 of file SimulationRunner.h.

### 9.26.2 Constructor & Destructor Documentation

#### 9.26.2.1 SimulationRunner()

```
SimulationRunner::SimulationRunner (
            unsigned int threads ) [explicit]
```

Creates a SimulationRunner which can use no more than the given number of threads to simulate Parameter combinations concurrently.

**Parameters**

| | |
|---|---|
| *threads* | Maximum number of threads that may be used for concurrent simulations. |

Definition at line 6 of file SimulationRunner.cpp.

### 9.26.3 Member Function Documentation

#### 9.26.3.1 getName()

```
string SimulationRunner::getName ( ) [override], [virtual]
```

Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented from Status.
Definition at line 14 of file SimulationRunner.cpp.

#### 9.26.3.2 getStatus()

```
string SimulationRunner::getStatus ( ) [override], [virtual]
```

Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented from Status.
Definition at line 18 of file SimulationRunner.cpp.

#### 9.26.3.3 getStatusBar()

```
string SimulationRunner::getStatusBar ( ) [override], [virtual]
```

Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented from Status.
Definition at line 22 of file SimulationRunner.cpp.

#### 9.26.3.4 runSimulations()

```
map< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, CmpVectorSharedParameter
> SimulationRunner::runSimulations (
            const set< vector< shared_ptr< Parameter >>, CmpVectorSharedParameter > & runs
)  [virtual]
```

Simulates the given Parameter combinations concurrently and returns their respective results.
Basically calls Multithreaded::runMultithreadedFunctions which uses the ThreadPool pattern to parallelize the execution of work.

**Parameters**

| | |
|---|---|
| *runs* | Set of Parameter combinations to be simulated. |

**Returns**

A map which maps the given Parameter combinations to their respective result directory and runIds.

Definition at line 10 of file SimulationRunner.cpp.

#### 9.26.3.5 work()

```
pair<filesystem::path, set<runId> > SimulationRunner::work (
            vector< shared_ptr< Parameter >> run )  [override], [private], [pure virtual]
```

Deals with the simulation of a single Parameter combination.
Overrides Multithreaded::work and therefore can be executed concurrently.

**Parameters**

| | |
|---|---|
| *run* | Parameter combination to be simulated. |

**Returns**

A pair containing a path to the result directory and a set of runIds identifying the respective simulation runs.

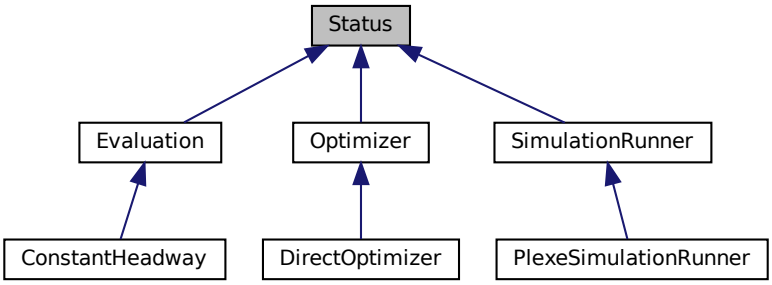
Implemented in PlexeSimulationRunner.
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/runner/SimulationRunner.h
- /home/runner/work/simopticon/simopticon/src/runner/SimulationRunner.cpp

## 9.27 Status Class Reference

An interface defining functions for status updates on configuration and progress of a class.
`#include "Status.h"`

Inheritance diagram for Status:




## Public Member Functions

- virtual string getName ()

  *Returns a string representing the name of the implementing component in natural language.*
- virtual string getStatus ()

  *Returns a string representing the current state of the implementing component.*
- virtual string getStatusBar ()

  *Returns a string representing the current progress of the calculations of the implementing component.*

## Static Protected Attributes

- static const string NO_STATUS_SUPPORT = "Component doesn't support status updates!"

  *Default message returned by getStatus and getStatusBar if the implementing class does not override the respective function.*
- static const string NO_NAME = "No name specified"

  *Default message returned by getName if the implementing class does not override the function.*

### 9.27.1 Detailed Description

An interface defining functions for status updates on configuration and progress of a class.
Used for creation of a StatusBar. Overriding the defined methods is not mandatory but recommended.
Definition at line 18 of file Status.h.

### 9.27.2 Member Function Documentation

#### 9.27.2.1 getName()

```
string Status::getName ( )  [virtual]
```

Returns a string representing the name of the implementing component in natural language.

**Returns**

A string containing the name of the component.

Reimplemented in SimulationRunner, PlexeSimulationRunner, Optimizer, DirectOptimizer, Evaluation, and ConstantHeadway.
Definition at line 16 of file Status.cpp.
References NO_NAME.
Referenced by Evaluation::getName(), Optimizer::getName(), and StatusBar::printStatus().

#### 9.27.2.2 getStatus()

```
string Status::getStatus ( )  [virtual]
```

Returns a string representing the current state of the implementing component.
May contain values of class members or other meaningful information. The returned string is always visible in StatusBar.

**Returns**

A string containing the state of the component.

Reimplemented in SimulationRunner, PlexeSimulationRunner, Optimizer, DirectOptimizer, Evaluation, and ConstantHeadway.
Definition at line 8 of file Status.cpp.
References NO_STATUS_SUPPORT.
Referenced by Evaluation::getStatus(), Optimizer::getStatus(), and StatusBar::printStatus().

#### 9.27.2.3 getStatusBar()

```
string Status::getStatusBar ( )  [virtual]
```

Returns a string representing the current progress of the calculations of the implementing component.
The returned string is visible in StatusBar, when the component is actively calculating something. Must not exceed one console line!

**Returns**

A string containing the progress of a calculation.

Reimplemented in SimulationRunner, PlexeSimulationRunner, Optimizer, DirectOptimizer, Evaluation, and ConstantHeadway.
Definition at line 12 of file Status.cpp.
References NO_STATUS_SUPPORT.
Referenced by Evaluation::getStatusBar(), Optimizer::getStatusBar(), and StatusBar::updateStatus().

### 9.27.3 Member Data Documentation

#### 9.27.3.1 NO_NAME

```
const string Status::NO_NAME = "No name specified"  [static], [protected]
```

Default message returned by getName if the implementing class does not override the function.
Definition at line 27 of file Status.h.
Referenced by getName().

#### 9.27.3.2 NO_STATUS_SUPPORT

```
const string Status::NO_STATUS_SUPPORT = "Component doesn't support status updates!"  [static],
[protected]
```

Default message returned by getStatus and getStatusBar if the implementing class does not override the respective function.
Definition at line 23 of file Status.h.
Referenced by getStatus(), and getStatusBar().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/status/Status.h
- /home/runner/work/simopticon/simopticon/src/status/Status.cpp

## 9.28 StatusBar Class Reference

A class used to conduct command line output containing information about the state of the used Optimizer, SimulationRunner and Evaluation along with the found optima.
`#include "StatusBar.h"`

### Public Member Functions

- void updateStatus (Optimizer *opt, SimulationRunner *runner, Evaluation *eval, const pair< vector< shared_ptr< Parameter >>, functionValue > &currentVal, bool stepChanged=false, step currentStep=INIT)

  *Updates the output in the command line with gathered information from the used Optimizer, SimulationRunner and Evaluation.*

### Static Public Member Functions

- static void printResults (list< pair< vector< shared_ptr< Parameter >>, pair< functionValue, filesystem↩ ::path >>> top)

  *Prints the given Parameter combinations and respective values to command line.*

### Static Private Member Functions

- static void printResult (const vector< shared_ptr< Parameter >> &cords, functionValue optimum)

  *Prints the given result command line.*
- static void printStatus (Status *object)

  *Prints the Status of the given object to the command line using Status::getStatus.*

### Private Attributes

- pair< vector< shared_ptr< Parameter > >, functionValue > lastVal

  *Pair of Parameter combination and respective value used to discern if the best value has changed since the last call to updateStatus.*
- step lastStep = INIT

  *Step which the optimization was in when updateStatus was called the last time.*
- string lastStatus

  *Last values of the StatusBar output (excluding value returned by Status::getStatusBar)*

### Static Private Attributes

- static const string LARGE_DIVIDER = "\n\n" + string(70, '#') + "\n"

  *Large divider used to visibly divide two sections of content.*
- static const string SMALL_DIVIDER = string(70, '-') + "\n"

  *Small divider used to visibly divide two sections of content.*

### 9.28.1 Detailed Description

A class used to conduct command line output containing information about the state of the used Optimizer, SimulationRunner and Evaluation along with the found optima.
Definition at line 35 of file StatusBar.h.

### 9.28.2 Member Function Documentation

#### 9.28.2.1 printResult()

```
void StatusBar::printResult (
            const vector< shared_ptr< Parameter >> & cords,
            functionValue optimum )  [static], [private]
```

Prints the given result command line.

**Parameters**

| | |
|---|---|
| *cords* | Parameter combination of the given result. |
| *optimum* | Value of the given result. |

Definition at line 54 of file StatusBar.cpp.
References Parameter::getConfig(), Parameter::getUnit(), and Parameter::getVal().
Referenced by updateStatus().

#### 9.28.2.2 printResults()

```
void StatusBar::printResults (
            list< pair< vector< shared_ptr< Parameter >>, pair< functionValue, filesystem↩
::path >>> top )  [static]
```

Prints the given Parameter combinations and respective values to command line.

**Parameters**

| | |
|---|---|
| *top* | List of Parameter combinations and respective values to be printed. |

Definition at line 73 of file StatusBar.cpp.
References LARGE_DIVIDER.

#### 9.28.2.3 printStatus()

```
void StatusBar::printStatus (
            Status * object )  [static], [private]
```

Prints the Status of the given object to the command line using Status::getStatus.

**Parameters**

| | |
|---|---|
| *object* | Object that inherits from Status and whose state is being printed. |

Definition at line 68 of file StatusBar.cpp.
References Status::getName(), and Status::getStatus().
Referenced by updateStatus().

#### 9.28.2.4 updateStatus()

```
void StatusBar::updateStatus (
            Optimizer * opt,
            SimulationRunner * runner,
            Evaluation * eval,
            const pair< vector< shared_ptr< Parameter >>, functionValue > & currentVal,
            bool stepChanged = false,
            step currentStep = INIT )
```

Updates the output in the command line with gathered information from the used Optimizer, SimulationRunner and Evaluation.

If the current optimum or the step the optimization is in has changed since the last call, the whole output is printed again. Otherwise only the progress of the active component obtained by Status::getStatusBar is updated.

**Parameters**

| | |
|---|---|
| *opt* | Pointer to Optimizer used in optimization. |
| *runner* | Pointer to SimulationRunner used in optimization. |
| *eval* | Pointer to Evaluation used in optimization. |
| *currentVal* | Parameter combination and respective value of the current optimum. |
| *stepChanged* | Boolean defining whether the current step has changed since the last call. |
| *currentStep* | Current step the optimization is in. |

Definition at line 13 of file StatusBar.cpp.
References Status::getStatusBar(), LARGE_DIVIDER, lastStatus, lastStep, lastVal, printResult(), printStatus(), and SMALL_DIVIDER.

## 9.28.3 Member Data Documentation

### 9.28.3.1 LARGE_DIVIDER

```
const string StatusBar::LARGE_DIVIDER = "\n\n" + string(70, '#') + "\n"  [static], [private]
```

Large divider used to visibly divide two sections of content.
Definition at line 40 of file StatusBar.h.
Referenced by printResults(), and updateStatus().

### 9.28.3.2 lastStatus

```
string StatusBar::lastStatus  [private]
```

Last values of the StatusBar output (excluding value returned by Status::getStatusBar)
Definition at line 57 of file StatusBar.h.
Referenced by updateStatus().

### 9.28.3.3 lastStep

```
step StatusBar::lastStep = INIT  [private]
```

Step which the optimization was in when updateStatus was called the last time.
Definition at line 53 of file StatusBar.h.
Referenced by updateStatus().

### 9.28.3.4 lastVal

```
pair<vector<shared_ptr<Parameter> >, functionValue> StatusBar::lastVal  [private]
```

Pair of Parameter combination and respective value used to discern if the best value has changed since the last call to updateStatus.
Definition at line 49 of file StatusBar.h.
Referenced by updateStatus().

### 9.28.3.5 SMALL_DIVIDER

```
const string StatusBar::SMALL_DIVIDER = string(70, '-') + "\n"  [static], [private]
```

Small divider used to visibly divide two sections of content.
Definition at line 44 of file StatusBar.h.

Referenced by updateStatus().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/status/StatusBar.h
- /home/runner/work/simopticon/simopticon/src/status/StatusBar.cpp

## 9.29 Controller::stepstate Struct Reference

A struct keeping track of the currently running optimization step for StatusBar::updateStatus.

### Public Member Functions

- void next ()

  *Switches currentStep to the next step.*
- step get ()

  *Returns the value of currentStep.*

### Public Attributes

- bool stepChanged

  *Defines if currentStep has changed since the last call to get.*
- step currentStep = INIT

  *Current step the optimization is in.*

### 9.29.1 Detailed Description

A struct keeping track of the currently running optimization step for StatusBar::updateStatus.
Definition at line 51 of file Controller.h.

### 9.29.2 Member Function Documentation

#### 9.29.2.1 get()

```
step Controller::stepstate::get ( )  [inline]
```

Returns the value of currentStep.

**Returns**

The step that is currently run.

Definition at line 73 of file Controller.h.
References currentStep, and stepChanged.

#### 9.29.2.2 next()

```
void Controller::stepstate::next ( )  [inline]
```

Switches currentStep to the next step.
Definition at line 64 of file Controller.h.
References currentStep, and stepChanged.
Referenced by Controller::requestValues(), and Controller::run().

### 9.29.3 Member Data Documentation

#### 9.29.3.1 currentStep

```
step Controller::stepstate::currentStep = INIT
```
Current step the optimization is in.
Definition at line 59 of file Controller.h.
Referenced by get(), and next().

#### 9.29.3.2 stepChanged

```
bool Controller::stepstate::stepChanged
```
Defines if currentStep has changed since the last call to get.
Definition at line 55 of file Controller.h.
Referenced by get(), next(), and Controller::updateStatus().
The documentation for this struct was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/controller/Controller.h

## 9.30 StoppingCondition Class Reference

A class used for deciding whether the DIRECT should be stopped.
`#include "StoppingCondition.h"`

### Public Member Functions

- StoppingCondition (size_t evaluations=0, size_t hyrects=0, unsigned int minutes=0, functionValue accuracy=0, unsigned int accuracyIterations=0)

  *Creates a StoppingCondition with the given condition values.*
- StoppingCondition (json stopCon)

  *Creates a StoppingCondition based on the given json configuration.*
- void setStartNow ()

  *Sets END_TIME to be the current time plus mins.*
- bool evaluate (size_t evaluations, size_t hyrects, functionValue newBestVal)

  *Checks if any of the configured conditions is met for the given parameters.*
- unsigned int getIterationsSinceImprov () const

  *Returns the value of iterationsSinceImprov.*

### Private Member Functions

- bool updateAccuracy (functionValue newBestVal)

  *Checks if the current optimum improves the one saved in bestVal by more than ACCURACY.*

### Private Attributes

- const size_t NR_EVALUATIONS

  *Number of evaluations after which the optimization should stop.*
- const size_t NR_HYRECTS

  *Number of rectangles in the partition after which the optimization should stop.*
- time_point< system_clock, seconds > END_TIME

  *Point in time after which optimization should end.*
- const unsigned int mins

  *Number of minutes after which the optimization should stop.*
- bool time_eval

  *Defines whether the time condition should be used.*
- const functionValue ACCURACY

*Accuracy used in accuracy condition.*

- const unsigned int NR_ACCURACY_ITERATIONS

  *Number of iterations used in accuracy condition.*

- functionValue bestVal = INFINITY

  *Best value used to keep track of accuracy condition.*

- unsigned int iterationsSinceImprov = 0

  *Number of iterations since last improvement of the optimum used to keep track of accuracy condition.*

## 9.30.1 Detailed Description

A class used for deciding whether the DIRECT should be stopped.
Every conditions is optional and can be set in config. The optimization is stopped when one of the activated conditions is met.
Definition at line 19 of file StoppingCondition.h.

## 9.30.2 Constructor & Destructor Documentation

### 9.30.2.1 StoppingCondition() [1/2]

```
StoppingCondition::StoppingCondition (
            size_t evaluations = 0,
            size_t hyrects = 0,
            unsigned int minutes = 0,
            functionValue accuracy = 0,
            unsigned int accuracyIterations = 0 )  [explicit]
```

Creates a StoppingCondition with the given condition values.

**Parameters**

| | |
|---|---|
| *evaluations* | Number of evaluations after which the optimization should stop. |
| *hyrects* | Number of rectangles in the partition after which the optimization should stop. |
| *minutes* | Number of minutes after which the optimization should stop. |
| *accuracy* | Accuracy used in accuracy condition (see ACCURACY). |
| *accuracyIterations* | Number of iterations used in accuracy condition (see NR_ACCURACY_ITERATIONS). |

Definition at line 3 of file StoppingCondition.cpp.
References ACCURACY, mins, NR_ACCURACY_ITERATIONS, NR_EVALUATIONS, NR_HYRECTS, and time_↩eval.

### 9.30.2.2 StoppingCondition() [2/2]

```
StoppingCondition::StoppingCondition (
            json stopCon )  [explicit]
```

Creates a StoppingCondition based on the given json configuration.

**Parameters**

| | |
|---|---|
| *stopCon* | JSON object defining the condition values. |

Definition at line 15 of file StoppingCondition.cpp.
References StoppingCondition().
Referenced by StoppingCondition().

## 9.30.3 Member Function Documentation

### 9.30.3.1 evaluate()

```
bool StoppingCondition::evaluate (
            size_t evaluations,
            size_t hyrects,
            functionValue newBestVal )
```

Checks if any of the configured conditions is met for the given parameters.

**Parameters**

| | |
|---|---|
| *evaluations* | Number of evaluations conducted by the optimization. |
| *hyrects* | Number of rectangles in the current partition. |
| *newBestVal* | Value of the current optimum. |

**Returns**

A boolean defining whether none of the configured conditions is met (meaning whether the optimization should keep running).

Definition at line 25 of file StoppingCondition.cpp.
References ACCURACY, NR_ACCURACY_ITERATIONS, NR_EVALUATIONS, NR_HYRECTS, and updateAccuracy().

### 9.30.3.2 getIterationsSinceImprov()

```
unsigned int StoppingCondition::getIterationsSinceImprov ( ) const
```

Returns the value of iterationsSinceImprov.

**Returns**

An integral representing the number of iterations since the best value improved by more than ACCURACY.

Definition at line 49 of file StoppingCondition.cpp.
References iterationsSinceImprov.

### 9.30.3.3 setStartNow()

```
void StoppingCondition::setStartNow ( )
```

Sets END_TIME to be the current time plus mins.
Definition at line 33 of file StoppingCondition.cpp.
References time_eval.

### 9.30.3.4 updateAccuracy()

```
bool StoppingCondition::updateAccuracy (
            functionValue newBestVal )  [private]
```

Checks if the current optimum improves the one saved in bestVal by more than ACCURACY.
If that is the case, iterationsSinceImprov is reset to zero and the current optimum is saved in bestVal. If not iterationsSinceImprov is increased.

**Parameters**

| | |
|---|---|
| *newBestVal* | Current optimum. |

**Returns**

A bool defining if the accuracy condition is met after the values where updated.

Definition at line 39 of file StoppingCondition.cpp.
References ACCURACY, bestVal, iterationsSinceImprov, and NR_ACCURACY_ITERATIONS.
Referenced by evaluate().

## 9.30.4 Member Data Documentation

### 9.30.4.1 ACCURACY

```
const functionValue StoppingCondition::ACCURACY  [private]
```

Accuracy used in accuracy condition.
When the bestVal has not changed more than ACCURACY after NR_ACCURACY_ITERATIONS iterations, the optimization is stopped.
Definition at line 48 of file StoppingCondition.h.
Referenced by StoppingCondition(), evaluate(), and updateAccuracy().

### 9.30.4.2 bestVal

```
functionValue StoppingCondition::bestVal = INFINITY  [private]
```

Best value used to keep track of accuracy condition.
When the bestVal has not changed more than ACCURACY after NR_ACCURACY_ITERATIONS iterations, the optimization is stopped.
Definition at line 58 of file StoppingCondition.h.
Referenced by updateAccuracy().

### 9.30.4.3 END_TIME

```
time_point<system_clock, seconds> StoppingCondition::END_TIME  [private]
```

Point in time after which optimization should end.
Calculated using time when setStartNow is called and mins.
Definition at line 34 of file StoppingCondition.h.

### 9.30.4.4 iterationsSinceImprov

```
unsigned int StoppingCondition::iterationsSinceImprov = 0  [private]
```

Number of iterations since last improvement of the optimum used to keep track of accuracy condition.
When the bestVal has not changed more than ACCURACY after NR_ACCURACY_ITERATIONS iterations, the optimization is stopped.
Definition at line 63 of file StoppingCondition.h.
Referenced by getIterationsSinceImprov(), and updateAccuracy().

### 9.30.4.5 mins

```
const unsigned int StoppingCondition::mins  [private]
```

Number of minutes after which the optimization should stop.
Definition at line 38 of file StoppingCondition.h.
Referenced by StoppingCondition().

#### 9.30.4.6 NR_ACCURACY_ITERATIONS

```
const unsigned int StoppingCondition::NR_ACCURACY_ITERATIONS  [private]
```

Number of iterations used in accuracy condition.
When the bestVal has not changed more than ACCURACY after NR_ACCURACY_ITERATIONS iterations, the optimization is stopped.
Definition at line 53 of file StoppingCondition.h.
Referenced by StoppingCondition(), evaluate(), and updateAccuracy().

#### 9.30.4.7 NR_EVALUATIONS

```
const size_t StoppingCondition::NR_EVALUATIONS  [private]
```

Number of evaluations after which the optimization should stop.
Definition at line 24 of file StoppingCondition.h.
Referenced by StoppingCondition(), and evaluate().

#### 9.30.4.8 NR_HYRECTS

```
const size_t StoppingCondition::NR_HYRECTS  [private]
```

Number of rectangles in the partition after which the optimization should stop.
Definition at line 28 of file StoppingCondition.h.
Referenced by StoppingCondition(), and evaluate().

#### 9.30.4.9 time_eval

```
bool StoppingCondition::time_eval  [private]
```

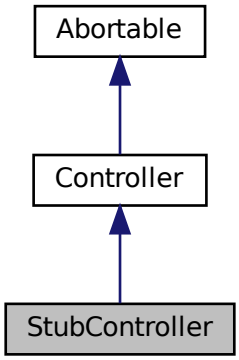
Defines whether the time condition should be used.
Definition at line 42 of file StoppingCondition.h.
Referenced by StoppingCondition(), and setStartNow().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/optimizer/direct/StoppingCondition.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/StoppingCondition.cpp

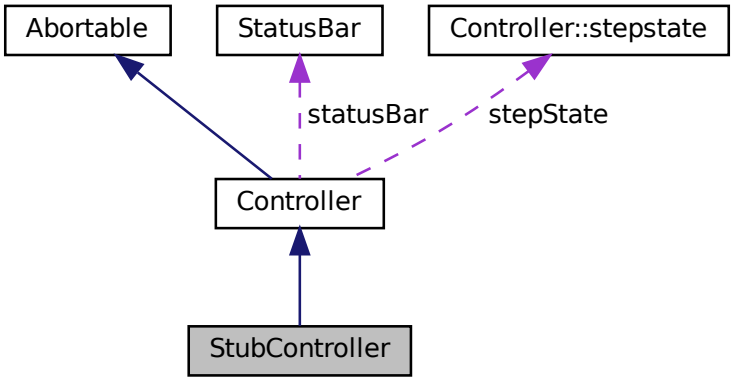
## 9.31 StubController Class Reference

A class that mocks behaviour of Controller.
`#include "StubController.h"`

Inheritance diagram for StubController:




Collaboration diagram for StubController:




## Public Member Functions

- StubController (const filesystem::path &configPath, const string &function)

  *Creates a StubController with the given config and function.*

## Private Member Functions

- map< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, CmpVectorSharedParameter > runSimulations (const set< vector< shared_ptr< Parameter >>, CmpVectorSharedParameter > &runs) override

  *Returns empty paths and runIds for each requested Parameter combination.*
- map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > evaluate (const map< vector< shared_ptr< Parameter >>, pair< filesystem::path, set< runId >>, CmpVectorSharedParameter > &simulationResults) override

*Evaluates the given Parameter combinations with f.*

- void removeOldResultfiles () override

  *Does nothing, since no simulations are run and therefore no result files are created.*

## Private Attributes

- const function< functionValue(vector< shared_ptr< Parameter >>)> f

  *Function to be optimized in the current optimization.*

## Static Private Attributes

- static map< string, function< functionValue(vector< shared_ptr< Parameter >>)> > functions

  *Map that contains the predefined functions quadratic, shekel5, shekel7, shekel10, branin, goldprice, camel6, shubert, hartman3 and hartman6.*

## Additional Inherited Members

### 9.31.1 Detailed Description

A class that mocks behaviour of Controller.
Instead of real simulations one of the predefined function in functions is being evaluated, when Controller::requestValues is called. To use StubController instead of Controller a second command line argument has to be passed containing the name of the function to be optimized. The name can be one of the following: quadratic, shekel5, shekel7, shekel10, branin, goldprice, camel6, shubert, hartman3 or hartman6. For more information on all but the first function visit: https://www.sfu.ca/~ssurjano/optimization.html
Definition at line 17 of file StubController.h.

### 9.31.2 Constructor & Destructor Documentation

#### 9.31.2.1 StubController()

```
StubController::StubController (
            const filesystem::path & configPath,
            const string & function )
```

Creates a StubController with the given config and function.

**Parameters**

| | |
|---|---|
| *configPath* | Path to the main config. Chosen by first command line argument. |
| *function* | Name of the function to be used. Chosen by second command line argument. |

Definition at line 119 of file StubController.cpp.
References StubController(), f, and functions.
Referenced by StubController().

### 9.31.3 Member Function Documentation

#### 9.31.3.1 evaluate()

```
map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > Stub↩
Controller::evaluate (
            const map< vector< shared_ptr< Parameter >>, pair< filesystem::path, set< runId
>>, CmpVectorSharedParameter > & simulationResults )  [override], [private], [virtual]
```

Evaluates the given Parameter combinations with f.

**Parameters**

| | |
|---|---|
| *simulationResults* | Map which maps Parameter combinations to empty results (see runSimulations). |

**Returns**

A Map which maps the given Parameter combinations to the respective value of f.

Reimplemented from Controller.
Definition at line 132 of file StubController.cpp.

#### 9.31.3.2 removeOldResultfiles()

```
void StubController::removeOldResultfiles ( )  [override], [private], [virtual]
```

Does nothing, since no simulations are run and therefore no result files are created.
Reimplemented from Controller.
Definition at line 141 of file StubController.cpp.

#### 9.31.3.3 runSimulations()

```
map< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, CmpVectorSharedParameter
> StubController::runSimulations (
            const set< vector< shared_ptr< Parameter >>, CmpVectorSharedParameter > & runs
)  [override], [private], [virtual]
```

Returns empty paths and runIds for each requested Parameter combination.

**Parameters**

| | |
|---|---|
| *runs* | Parameter combination to be simulated. |

**Returns**

Map which maps the given Parameter combinations to empty paths and runIds.

Reimplemented from Controller.
Definition at line 124 of file StubController.cpp.

### 9.31.4 Member Data Documentation

#### 9.31.4.1 f

```
const function<functionValue(vector<shared_ptr<Parameter>>)> StubController::f  [private]
```

Function to be optimized in the current optimization.
One of the functions in functions.
Definition at line 28 of file StubController.h.
Referenced by StubController().

#### 9.31.4.2 functions

```
map<string, function<functionValue(vector<shared_ptr<Parameter>>)> > StubController::functions
[static], [private]
```

Map that contains the predefined functions quadratic, shekel5, shekel7, shekel10, branin, goldprice, camel6, shubert, hartman3 and hartman6.

For more information on all but the first function visit: https://www.sfu.ca/~ssurjano/optimization.html
Definition at line 23 of file StubController.h.
Referenced by StubController().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/controller/StubController.h
- /home/runner/work/simopticon/simopticon/src/controller/StubController.cpp

## 9.32 ThreadsafeQueue< Key > Class Template Reference

A container class of a queue that is safe for concurrent access of different threads.
`#include "ThreadsafeQueue.h"`

### Public Member Functions

- void push (Key val)

  *Adds the given value to safeQueue.*
- pair< Key, bool > pop ()

  *Returns the first element of the queue.*
- size_t getStartSize ()

  *Returns the value of startSize.*
- size_t getSize ()

  *Returns current size of the underlying queue structure.*

### Private Attributes

- queue< Key > safeQueue

  *The actual queue data structure.*
- mutex queueLock

  *Threadlock to avoid damage to safeQueue on concurrent access.*
- size_t startSize = 0

  *Number of elements in queue when push was called the last time.*

### 9.32.1 Detailed Description

**template<class Key>**
**class ThreadsafeQueue< Key >**

A container class of a queue that is safe for concurrent access of different threads.

**Template Parameters**

| | |
|---|---|
| *Key* | Type of elements in the contained queue. |

Definition at line 16 of file ThreadsafeQueue.h.

### 9.32.2 Member Function Documentation

#### 9.32.2.1 getSize()

```
template<class Key >
size_t ThreadsafeQueue< Key >::getSize ( )
```

Returns current size of the underlying queue structure.

**Returns**

A number representing the size of the queue.

#### 9.32.2.2 getStartSize()

```
template<class Key >
size_t ThreadsafeQueue< Key >::getStartSize ( )
```

Returns the value of startSize.

**Returns**

A number representing the number of tasks, when push was called last.

#### 9.32.2.3 pop()

```
template<class Key >
pair<Key, bool> ThreadsafeQueue< Key >::pop ( )
```

Returns the first element of the queue.
If the queue is empty, the second entry of the returned pair is false.

**Returns**

A pair containing an element of type Key and a boolean determining if access was successful.

#### 9.32.2.4 push()

```
template<class Key >
void ThreadsafeQueue< Key >::push (
            Key val )
```

Adds the given value to safeQueue.

**Parameters**

| | |
|---|---|
| *val* | Values to be added to queue. |

### 9.32.3 Member Data Documentation

#### 9.32.3.1 queueLock

```
template<class Key >
mutex ThreadsafeQueue< Key >::queueLock  [private]
```

Threadlock to avoid damage to safeQueue on concurrent access.
Definition at line 25 of file ThreadsafeQueue.h.

#### 9.32.3.2 safeQueue

```
template<class Key >
queue<Key> ThreadsafeQueue< Key >::safeQueue  [private]
```

The actual queue data structure.
Definition at line 21 of file ThreadsafeQueue.h.

#### 9.32.3.3 startSize

```
template<class Key >
size_t ThreadsafeQueue< Key >::startSize = 0  [private]
```

Number of elements in queue when push was called the last time.
Can be used for progress information.
Definition at line 30 of file ThreadsafeQueue.h.
The documentation for this class was generated from the following file:

- /home/runner/work/simopticon/simopticon/src/utils/ThreadsafeQueue.h

## 9.33 ValueMap Class Reference

A container managing a map data structure that maps Parameter combinations to their respective found values.
`#include "ValueMap.h"`

### Public Member Functions

- ValueMap (unsigned int topEntries=10)

  *Creates a ValueMap.*
- functionValue query (const vector< shared_ptr< Parameter >> &params)

  *Returns the value saved at the given Parameter combination.*
- void insert (const vector< shared_ptr< Parameter >> &params, functionValue val)

  *Adds the given Parameter combination and value to tba.*
- bool isKnown (const vector< shared_ptr< Parameter >> &cords)

  *Checks if a value has been recorded at the given Parameter combination.*
- bool isTopValue (const vector< shared_ptr< Parameter >> &cords)

  *Checks if the given Parameter combination is to be found in topVals.*
- const map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > & getValues ()

  *Returns the whole values member.*
- functionValue getMedian ()

  *Returns the median of all values using lowerValues and upperValues.*
- size_t getSize () const

  *Returns the number of inserted values.*
- list< pair< vector< shared_ptr< Parameter > >, functionValue > > getTopVals ()

  *Returns the best topEntries entries that are saved in topVals.*

### Private Member Functions

- void updateMap ()

  *Takes all values in tba, adds them to lowerValues or upperValues and inserts them into values.*
- void addValue (const pair< vector< shared_ptr< Parameter >>, functionValue > &val, set< functionValue *, CmpPtrFunctionvalue > &set)

  *Inserts a single value into values and into lowerValues or upperValues depending on set argument.*

### Private Attributes

- mutex operationsLock

  *Threadlock to avoid damage to the data structure when concurrent threads access it.*
- set< functionValue *, CmpPtrFunctionvalue > upperValues

  *Greater half of the values in values.*
- set< functionValue *, CmpPtrFunctionvalue > lowerValues

  *Lesser half of the values in values.*

- const unsigned int topEntries

  *Number of entries to be printed as best values at the end of the optimization process.*
- set< pair< const vector< shared_ptr< Parameter > >, functionValue >, CmpPairVectorSharedParameterFunctionvalue > topVals

  *Set of pairs of the best Parameter combinations and their respective values.*
- map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > values

  *Actual map that contains Parameter combinations and their respective values.*
- list< pair< vector< shared_ptr< Parameter > >, functionValue > > tba

  *Entries that have been added since last updateMap.*

## 9.33.1 Detailed Description

A container managing a map data structure that maps Parameter combinations to their respective found values.
The class manages concurrent access using the operationsLock. Running median calculation is supported by using sets upperValues and lowerValues. Values are inserted into the data structure at once when updateMap is called. Before that they are saved in tba to avoid unnecessary costly insertion operations.
Definition at line 26 of file ValueMap.h.

## 9.33.2 Constructor & Destructor Documentation

### 9.33.2.1 ValueMap()

```
ValueMap::ValueMap (
            unsigned int topEntries = 10 )  [explicit]
```

Creates a ValueMap.

**Parameters**

| | |
|---|---|
| *topEntries* | Value to be assigned to topEntries. |

Definition at line 7 of file ValueMap.cpp.
References topEntries.

## 9.33.3 Member Function Documentation

### 9.33.3.1 addValue()

```
void ValueMap::addValue (
            const pair< vector< shared_ptr< Parameter >>, functionValue > & val,
            set< functionValue *, CmpPtrFunctionvalue > & set )  [private]
```

Inserts a single value into values and into lowerValues or upperValues depending on *set* argument.

**Parameters**

| | |
|---|---|
| *val* | Parameter combination and respective value to be inserted. |
| *set* | Set that value is inserted in. Either lowerValues or upperValues. |

Definition at line 45 of file ValueMap.cpp.
References topEntries, topVals, and values.
Referenced by updateMap().

#### 9.33.3.2 getMedian()

```
functionValue ValueMap::getMedian ( )
```

Returns the median of all values using lowerValues and upperValues.
If no values have been added, 0 is returned. Triggers updateMap.

**Returns**

A value representing the median of all values.

Definition at line 92 of file ValueMap.cpp.
References getSize(), lowerValues, operationsLock, updateMap(), and upperValues.

#### 9.33.3.3 getSize()

```
size_t ValueMap::getSize ( ) const
```

Returns the number of inserted values.
Values in tba are included.

**Returns**

An integral representing the number of inserted values.

Definition at line 104 of file ValueMap.cpp.
References tba, and values.
Referenced by getMedian(), and isKnown().

#### 9.33.3.4 getTopVals()

```
list< pair< vector< shared_ptr< Parameter > >, functionValue > > ValueMap::getTopVals ( )
```

Returns the best topEntries entries that are saved in topVals.
Triggers updateMap.

**Returns**

A list of the best topEntries Parameter combinations and their respective values.

Definition at line 108 of file ValueMap.cpp.
References topVals, and updateMap().

#### 9.33.3.5 getValues()

```
const map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > &
ValueMap::getValues ( )
```

Returns the whole values member.
Triggers updateMap.

**Returns**

A map reference to values.

Definition at line 129 of file ValueMap.cpp.
References updateMap(), and values.

#### 9.33.3.6 insert()

```
void ValueMap::insert (
            const vector< shared_ptr< Parameter >> & params,
            functionValue val )
```

Adds the given Parameter combination and value to tba.

**Parameters**

| | |
|---|---|
| *params* | Parameter combination to be added. |
| *val* | Value to be added. |

Definition at line 77 of file ValueMap.cpp.
References tba.

#### 9.33.3.7 isKnown()

```
bool ValueMap::isKnown (
            const vector< shared_ptr< Parameter >> & cords )
```

Checks if a value has been recorded at the given Parameter combination.
Triggers updateMap.

**Parameters**

| | |
|---|---|
| *cords* | Parameter combination that is checked. |

**Returns**

A boolean value that represents if the value is known.

Definition at line 81 of file ValueMap.cpp.
References getSize(), operationsLock, updateMap(), and values.

#### 9.33.3.8 isTopValue()

```
bool ValueMap::isTopValue (
            const vector< shared_ptr< Parameter >> & cords )
```

Checks if the given Parameter combination is to be found in topVals.
Triggers updateMap.

**Parameters**

| | |
|---|---|
| *cords* | Parameter combination that is checked. |

**Returns**

A boolean value that represents if the value is one of the best topEntries entries in values.

Definition at line 113 of file ValueMap.cpp.
References Parameter::getVal(), topVals, and updateMap().

#### 9.33.3.9 query()

```
functionValue ValueMap::query (
            const vector< shared_ptr< Parameter >> & params )
```

Returns the value saved at the given Parameter combination.
If no value is present, an exception is thrown. Triggers updateMap.

**Parameters**

| | |
|---|---|
| *params* | Parameter combination to which the value is requested. |

**Returns**

The value saved in values at the given Parameter combination.

Definition at line 66 of file ValueMap.cpp.
References operationsLock, updateMap(), and values.

#### 9.33.3.10 updateMap()

`void ValueMap::updateMap ( ) [private]`
Takes all values in tba, adds them to lowerValues or upperValues and inserts them into values.
lowerValues and upperValues are sorted as is required by their constraints. Afterwards tba is cleared.
Definition at line 10 of file ValueMap.cpp.
References addValue(), lowerValues, operationsLock, tba, and upperValues.
Referenced by getMedian(), getTopVals(), getValues(), isKnown(), isTopValue(), and query().

### 9.33.4 Member Data Documentation

#### 9.33.4.1 lowerValues

`set<functionValue *, CmpPtrFunctionvalue> ValueMap::lowerValues [private]`
Lesser half of the values in values.
Same size as or one element less than upperValues.
Definition at line 40 of file ValueMap.h.
Referenced by getMedian(), and updateMap().

#### 9.33.4.2 operationsLock

`mutex ValueMap::operationsLock [private]`
Threadlock to avoid damage to the data structure when concurrent threads access it.
Definition at line 31 of file ValueMap.h.
Referenced by getMedian(), isKnown(), query(), and updateMap().

#### 9.33.4.3 tba

`list<pair<vector<shared_ptr<Parameter> >, functionValue> > ValueMap::tba [private]`
Entries that have been added since last updateMap.
Will be inserted into values, upperValues and lowerValues when updateMap is called.
Definition at line 62 of file ValueMap.h.
Referenced by getSize(), insert(), and updateMap().

#### 9.33.4.4 topEntries

`const unsigned int ValueMap::topEntries [private]`
Number of entries to be printed as best values at the end of the optimization process.
Can be configured in main config.
Definition at line 46 of file ValueMap.h.
Referenced by ValueMap(), and addValue().

#### 9.33.4.5 topVals

`set<pair<const vector<shared_ptr<Parameter> >, functionValue>, CmpPairVectorSharedParameterFunctionvalue> ValueMap::topVals [private]`

Set of pairs of the best Parameter combinations and their respective values.
Contains not more than topEntries entries.
Definition at line 51 of file ValueMap.h.
Referenced by addValue(), getTopVals(), and isTopValue().

#### 9.33.4.6 upperValues

`set<functionValue *, CmpPtrFunctionvalue> ValueMap::upperValues [private]`
Greater half of the values in values.
Same size as or one element more than lowerValues.
Definition at line 36 of file ValueMap.h.
Referenced by getMedian(), and updateMap().

#### 9.33.4.7 values

`map<vector<shared_ptr<Parameter> >, functionValue, CmpVectorSharedParameter> ValueMap::values [private]`
Actual map that contains Parameter combinations and their respective values.
Definition at line 56 of file ValueMap.h.
Referenced by addValue(), getSize(), getValues(), isKnown(), and query().
The documentation for this class was generated from the following files:

- /home/runner/work/simopticon/simopticon/src/controller/ValueMap.h
- /home/runner/work/simopticon/simopticon/src/controller/ValueMap.cpp

# Chapter 10

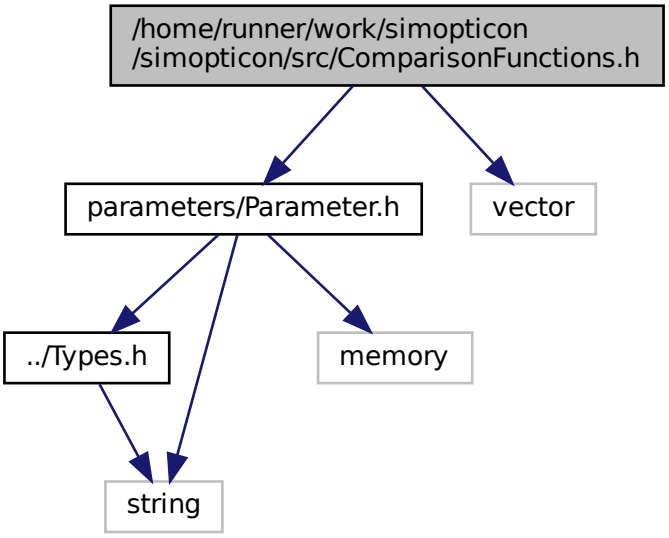
# File Documentation

## 10.1 /home/runner/work/simopticon/simopticon/src/ComparisonFunctions.h File Reference

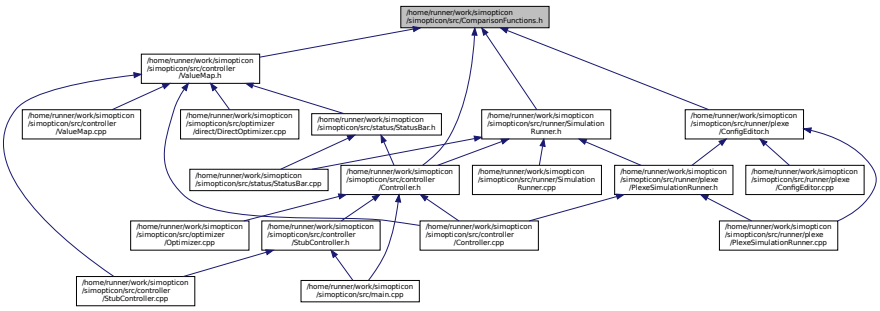
In this file comparison functions are defined which should be used across the whole framework.

```
#include "parameters/Parameter.h"
#include <vector>
```

Include dependency graph for ComparisonFunctions.h:



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




## Classes

- struct CmpVectorSharedParameter

  *This struct implements the comparison of two vectors of Parameter references.*
- struct CmpPtrFunctionvalue

  *This struct implements the comparison of two pointers to function values.*
- struct CmpPairVectorSharedParameterFunctionvalue

  *This struct implements the comparison of two pairs of Parameter combination and function value.*

### 10.1.1 Detailed Description

In this file comparison functions are defined which should be used across the whole framework.
They can be used to order elements in STL containers.

## 10.2 /home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/constant_headway.py File Reference

A Python script providing functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap.

## Functions

- np.float128 constant_headway.get_last_value (pd.DataFrame df)

  *Returns the last value of the numpy array located in the first row of the given DataFrame in field 'vecvalue'.*
- np.float128 constant_headway.get_constant_headway (list run_ids)

  *Calculates a value rating the mean deviation of all vehicles from the pre-defined gap.*
- list constant_headway.multithreaded (int threads, str directory, list run_ids)

  *Runs get_constant_headway concurrently for multiple simulation results with no more than the given number of threads.*

### 10.2.1 Detailed Description

A Python script providing functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap.
To achieve this, the OMNeT++ Python API omnetpp.scave is used. Multithreading is introduced to speed up the processing of multiple evaluations.

### 10.2.2 Function Documentation

#### 10.2.2.1 get_constant_headway()

```
np.float128 constant_headway.get_constant_headway (
            list run_ids )
```

Calculates a value rating the mean deviation of all vehicles from the pre-defined gap.
It calculates the mean squared deviation of each vehicle from its pre-defined gap, adds that value up for each vehicle of a particular run and calculates the mean over all runs (i.e., all repetitions and scenarios).

**Parameters**

| | |
|---|---|
| *run_ids* | List of strings representing the OMNeT++ run ids of all runs to be evaluated. |

**Returns**

A longfloat rating the deviation from the pre-defined gap.

**Bug** Running mean calculation over vectors using omnetpp.scave does not work correctly!

Definition at line 36 of file constant_headway.py.

#### 10.2.2.2 get_last_value()

```
np.float128 constant_headway.get_last_value (
            pd.DataFrame df )
```

Returns the last value of the numpy array located in the first row of the given DataFrame in field 'vecvalue'.

**Parameters**

| | |
|---|---|
| *df* | A DataFrame containing a recorded vector of simulation data. |

**Returns**

The longfloat at the last vector position in the first row of the DataFrame.

Definition at line 25 of file constant_headway.py.

#### 10.2.2.3 multithreaded()

```
list constant_headway.multithreaded (
            int threads,
            str directory,
            list run_ids )
```

Runs get_constant_headway concurrently for multiple simulation results with no more than the given number of threads.
This is the function actually called by ConstantHeadway.

**Parameters**

| | |
|---|---|
| *threads* | Maximum number of threads to be used for concurrent execution. |
| *directory* | A path to the directory directly or indirectly containing all result files that are to be evaluated. |
| *run_ids* | A list of lists of strings where each list of strings contains all OMNeT++ run ids of the runs conducted for one Parameter combination |

**Returns**

A list of longfloats representing the rating of the given simulation runs.

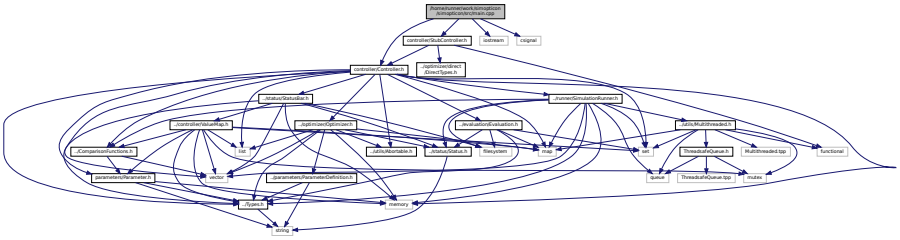
Definition at line 78 of file constant_headway.py.

## 10.3 /home/runner/work/simopticon/simopticon/src/main.cpp File Reference

Definition of the main function running the *Simopticon* framework.

```
#include "controller/Controller.h"
#include "controller/StubController.h"
#include <iostream>
#include <csignal>
```

Include dependency graph for main.cpp:



### Functions

- void interruptHandler ([[maybe_unused]] int s)

  *Handler routine for SIGINT signal which calls Controller::abort and sets the new handler of SIGINT to the default (instant interrupt of the software).*

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

  *Checks correct command line input and registers interrupt handler for SIGINT signal.*

### Variables

- unique_ptr< Controller > ctr

  *Reference to the Controller that is running the optimization.*

### 10.3.1 Detailed Description

Definition of the main function running the *Simopticon* framework.

### 10.3.2 Function Documentation

#### 10.3.2.1 interruptHandler()

```
void interruptHandler (
            [[maybe_unused]] int s )
```

Handler routine for SIGINT signal which calls Controller::abort and sets the new handler of SIGINT to the default (instant interrupt of the software).

**Parameters**

| | |
|---|---|
| *s* | Necessary parameter for interrupt handlers (unused). |

**Todo** Make interrupt handling independent from OS - currently only Systems using POSIX signals are supported.

Definition at line 24 of file main.cpp.

#### 10.3.2.2 main()

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

Checks correct command line input and registers interrupt handler for SIGINT signal.
Instantiates Controller or StubController and kicks of the optimization using Controller::run.

**Parameters**

| | |
|---|---|
| *argc* | Number of command line arguments. |
| *argv* | Array of command line arguments. |

**Returns**

Status code.

Definition at line 36 of file main.cpp.

### 10.3.3 Variable Documentation

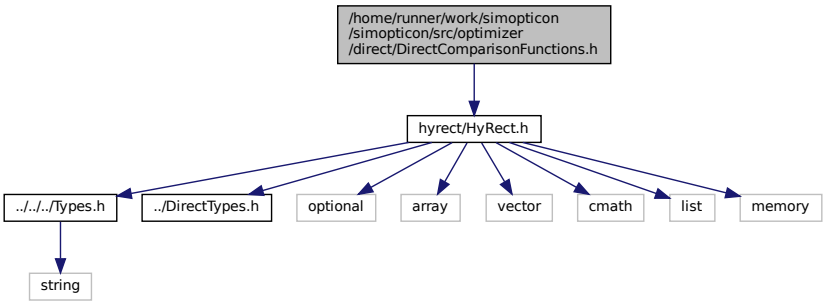
#### 10.3.3.1 ctr

```
unique_ptr<Controller> ctr
```

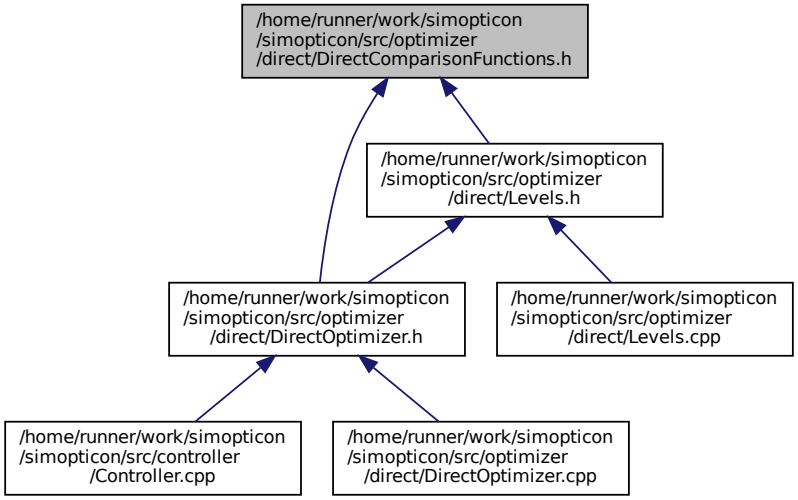
Reference to the Controller that is running the optimization.
Definition at line 17 of file main.cpp.

## 10.4 /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectComparisonFunctions.h File Reference

In this file comparison functions are defined which are used in the direct module.
`#include "hyrect/HyRect.h"`
Include dependency graph for DirectComparisonFunctions.h:



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




## Classes

- struct CmpSharedHyrect

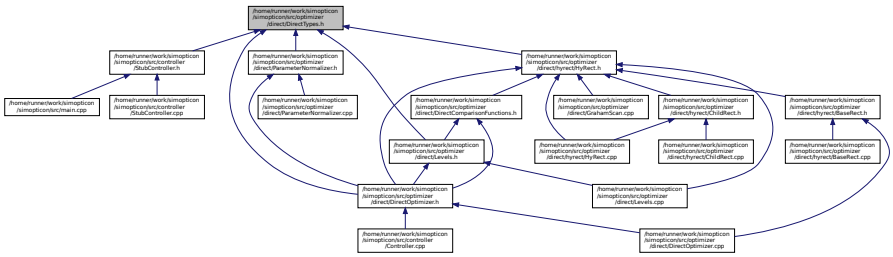
  *This struct implements the comparison of two shared pointers to HyRect instances.*

### 10.4.1 Detailed Description

In this file comparison functions are defined which are used in the direct module.
They can be used to order elements in STL containers.

## 10.5 /home/runner/work/simopticon/simopticon/src/optimizer/direct/↩ DirectTypes.h File Reference

In this file types are defined which are in the direct module.
This graph shows which files directly or indirectly include this file:



## Typedefs

- typedef unsigned int depth

  *An integral type used for representing the depth of a HyRect in the partition tree.*
- typedef unsigned char dimension

*An integral type used for representing a dimension of the search space.*

- typedef long double dirCoordinate

  *A floating point type used for representing one coordinate in the hypercube search space.*

### 10.5.1 Detailed Description

In this file types are defined which are in the direct module.

### 10.5.2 Typedef Documentation

#### 10.5.2.1 depth

```
typedef unsigned int depth
```

An integral type used for representing the depth of a HyRect in the partition tree.
Definition at line 14 of file DirectTypes.h.

#### 10.5.2.2 dimension

```
typedef unsigned char dimension
```

An integral type used for representing a dimension of the search space.
Please note that the first dimension is represented by value 1, not 0.
Definition at line 20 of file DirectTypes.h.

#### 10.5.2.3 dirCoordinate

```
typedef long double dirCoordinate
```

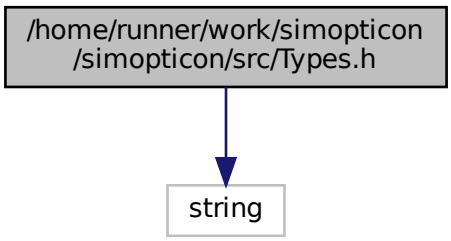
A floating point type used for representing one coordinate in the hypercube search space.
Definition at line 25 of file DirectTypes.h.

## 10.6 /home/runner/work/simopticon/simopticon/src/Types.h File Reference

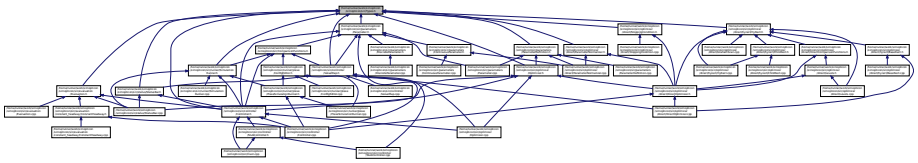
In this file types are defined which should be used across the whole framework.

```
#include <string>
```

Include dependency graph for Types.h:



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



## Typedefs

- typedef long double functionValue

  *A floating point type containing the value of an optimized function.*
- typedef double coordinate

  *A floating point type used to represent Parameter values.*
- typedef std::string runId

  *An identifier that makes different simulation runs in one result file folder distinguishable.*

### 10.6.1 Detailed Description

In this file types are defined which should be used across the whole framework.

### 10.6.2 Typedef Documentation

#### 10.6.2.1 coordinate

```
typedef double coordinate
```

A floating point type used to represent Parameter values.
Definition at line 21 of file Types.h.

#### 10.6.2.2 functionValue

```
typedef long double functionValue
```

A floating point type containing the value of an optimized function.
Definition at line 16 of file Types.h.

#### 10.6.2.3 runId

```
typedef std::string runId
```

An identifier that makes different simulation runs in one result file folder distinguishable.
Uniqueness is not being asserted.
Definition at line 27 of file Types.h.

# Index