

Simopticon

1.0

Generated by Doxygen 1.9.1



<b>1 Documentation</b>	<b>1</b>
1.1 Overview	1
1.2 Setup	2
1.2.1 Requirements	2
1.2.2 Installation	2
1.3 Usage	3
1.3.1 Configuration	3
1.3.2 Optimization	3
1.4 Extension	4
1.4.1 Development	4
1.4.2 Integration	5
<b>2 Todo List</b>	<b>7</b>
<b>3 Bug List</b>	<b>9</b>
<b>4 Module Index</b>	<b>11</b>
4.1 Modules	11
<b>5 Hierarchical Index</b>	<b>13</b>
5.1 Class Hierarchy	13
<b>6 Class Index</b>	<b>15</b>
6.1 Class List	15
<b>7 File Index</b>	<b>17</b>
7.1 File List	17
<b>8 Module Documentation</b>	<b>19</b>
8.1 controller	19
8.1.1 Detailed Description	19
8.1.2 Variable Documentation	19
8.1.2.1 stepState	19
8.2 direct	19
8.2.1 Detailed Description	20
8.2.2 Enumeration Type Documentation	20
8.2.2.1 level	21
8.3 plexe	21
8.3.1 Detailed Description	21
8.4 constant_headway	21
8.4.1 Detailed Description	22
8.5 parameters	22
8.5.1 Detailed Description	22
8.6 evaluation	22
8.6.1 Detailed Description	23

8.7 runner	23
8.7.1 Detailed Description	23
8.8 status	23
8.8.1 Detailed Description	24
8.8.2 Enumeration Type Documentation	24
8.8.2.1 step	24
8.9 optimizer	24
8.9.1 Detailed Description	24
8.10 hyrect	24
8.10.1 Detailed Description	25
8.10.2 Enumeration Type Documentation	25
8.10.2.1 position	25
8.11 utils	25
8.11.1 Detailed Description	26
<b>9 Class Documentation</b>	<b>27</b>
9.1 Abortable Class Reference	27
9.1.1 Detailed Description	27
9.1.2 Member Function Documentation	28
9.1.2.1 abort()	28
9.1.3 Member Data Documentation	28
9.1.3.1 aborted	28
9.2 BaseRect Class Reference	28
9.2.1 Detailed Description	29
9.2.2 Constructor & Destructor Documentation	29
9.2.2.1 BaseRect()	29
9.2.3 Member Function Documentation	29
9.2.3.1 getSamplingVertices()	29
9.3 ChildRect Class Reference	30
9.3.1 Detailed Description	31
9.3.2 Constructor & Destructor Documentation	31
9.3.2.1 ChildRect()	31
9.3.3 Member Function Documentation	31
9.3.3.1 getSamplingVertices()	31
9.3.3.2 operator==()	31
9.3.4 Member Data Documentation	32
9.3.4.1 parent	32
9.4 CmpPairVectorSharedParameterFunctionvalue Struct Reference	32
9.4.1 Detailed Description	32
9.4.2 Member Function Documentation	32
9.4.2.1 operator()()	32
9.5 CmpPtrFunctionvalue Struct Reference	33

9.5.1 Detailed Description . . . . .	33
9.5.2 Member Function Documentation . . . . .	33
9.5.2.1 operator()() . . . . .	33
9.6 CmpSharedHyrect Struct Reference . . . . .	33
9.6.1 Detailed Description . . . . .	34
9.6.2 Member Function Documentation . . . . .	34
9.6.2.1 operator()() . . . . .	34
9.7 CmpVectorSharedParameter Struct Reference . . . . .	34
9.7.1 Detailed Description . . . . .	34
9.7.2 Member Function Documentation . . . . .	34
9.7.2.1 operator()() . . . . .	35
9.8 CommandLine Class Reference . . . . .	35
9.8.1 Detailed Description . . . . .	35
9.8.2 Member Function Documentation . . . . .	35
9.8.2.1 exec() . . . . .	35
9.9 ConfigEditor Class Reference . . . . .	36
9.9.1 Detailed Description . . . . .	36
9.9.2 Constructor & Destructor Documentation . . . . .	37
9.9.2.1 ConfigEditor() . . . . .	37
9.9.3 Member Function Documentation . . . . .	37
9.9.3.1 createConfig() . . . . .	37
9.9.3.2 deleteConfig() . . . . .	37
9.9.3.3 getConfigPath() . . . . .	38
9.9.3.4 getControllerOption() . . . . .	38
9.9.3.5 getDir() . . . . .	38
9.9.3.6 getResultPath() . . . . .	38
9.9.3.7 replaceOption() [1/2] . . . . .	39
9.9.3.8 replaceOption() [2/2] . . . . .	39
9.9.3.9 setResultFiles() . . . . .	39
9.9.4 Member Data Documentation . . . . .	40
9.9.4.1 CONFIG . . . . .	40
9.9.4.2 CONTROLLER . . . . .	40
9.9.4.3 DIR . . . . .	40
9.9.4.4 RESULTS . . . . .	40
9.10 ConstantHeadway Class Reference . . . . .	40
9.10.1 Detailed Description . . . . .	42
9.10.2 Constructor & Destructor Documentation . . . . .	42
9.10.2.1 ConstantHeadway() . . . . .	42
9.10.3 Member Function Documentation . . . . .	42
9.10.3.1 getName() . . . . .	42
9.10.3.2 getStatus() . . . . .	43
9.10.3.3 getStatusBar() . . . . .	43

9.10.3.4 processOutput() [1/2]	43
9.10.3.5 processOutput() [2/2]	43
9.10.3.6 secureValue()	44
9.10.4 Member Data Documentation	44
9.10.4.1 NR_THREADS	44
9.10.4.2 usedThreads	44
9.11 ContinuousParameter Class Reference	44
9.11.1 Detailed Description	45
9.11.2 Constructor & Destructor Documentation	46
9.11.2.1 ContinuousParameter() [1/2]	46
9.11.2.2 ContinuousParameter() [2/2]	46
9.11.3 Member Function Documentation	46
9.11.3.1 getVal()	46
9.11.3.2 setVal()	46
9.11.4 Member Data Documentation	48
9.11.4.1 val	48
9.12 Controller Class Reference	48
9.12.1 Detailed Description	50
9.12.2 Constructor & Destructor Documentation	50
9.12.2.1 Controller()	50
9.12.3 Member Function Documentation	50
9.12.3.1 abort()	51
9.12.3.2 evaluate()	51
9.12.3.3 getValueMap()	51
9.12.3.4 removeOldResultfiles()	51
9.12.3.5 requestValues()	51
9.12.3.6 run()	52
9.12.3.7 runSimulations()	52
9.12.3.8 updateStatus()	52
9.12.4 Member Data Documentation	52
9.12.4.1 evaluation	53
9.12.4.2 keepFiles	53
9.12.4.3 optimizer	53
9.12.4.4 runner	53
9.12.4.5 statusBar	53
9.12.4.6 statusInterval	53
9.12.4.7 topResults	53
9.12.4.8 valueMap	53
9.13 DirectOptimizer Class Reference	54
9.13.1 Detailed Description	56
9.13.2 Constructor & Destructor Documentation	56
9.13.2.1 DirectOptimizer()	56

9.13.3 Member Function Documentation	56
9.13.3.1 addActiveRects()	56
9.13.3.2 estimatedValue()	56
9.13.3.3 getName()	57
9.13.3.4 getPartitionSize()	57
9.13.3.5 getStatus()	57
9.13.3.6 getStatusBar()	57
9.13.3.7 getValues()	58
9.13.3.8 optimalRectangles()	58
9.13.3.9 removeActiveRects()	58
9.13.3.10 runOptimization()	59
9.13.3.11 saveProgress()	59
9.13.3.12 saveValues()	59
9.13.4 Member Data Documentation	59
9.13.4.1 activeRects	59
9.13.4.2 D	59
9.13.4.3 iterations	60
9.13.4.4 level	60
9.13.4.5 normalizer	60
9.13.4.6 printValues	60
9.13.4.7 stopCon	60
9.13.4.8 trackProgress	60
9.14 DiscreteParameter Class Reference	60
9.14.1 Detailed Description	62
9.14.2 Constructor & Destructor Documentation	62
9.14.2.1 DiscreteParameter() [1/2]	62
9.14.2.2 DiscreteParameter() [2/2]	62
9.14.3 Member Function Documentation	63
9.14.3.1 getOffset()	63
9.14.3.2 getStep()	63
9.14.3.3 getTimes()	63
9.14.3.4 getVal()	63
9.14.3.5 setTimes()	63
9.14.3.6 setVal()	64
9.14.4 Member Data Documentation	64
9.14.4.1 offset	64
9.14.4.2 step	64
9.14.4.3 times	64
9.15 Evaluation Class Reference	65
9.15.1 Detailed Description	66
9.15.2 Member Function Documentation	66
9.15.2.1 getName()	66

9.15.2.2 <a href="#">getStatus()</a> . . . . .	66
9.15.2.3 <a href="#">getStatusBar()</a> . . . . .	66
9.15.2.4 <a href="#">processOutput()</a> [1/2] . . . . .	66
9.15.2.5 <a href="#">processOutput()</a> [2/2] . . . . .	67
9.16 <a href="#">GrahamScan Class Reference</a> . . . . .	67
9.16.1 Detailed Description . . . . .	67
9.16.2 Member Function Documentation . . . . .	67
9.16.2.1 <a href="#">scan()</a> . . . . .	68
9.17 <a href="#">HyRect Class Reference</a> . . . . .	68
9.17.1 Detailed Description . . . . .	69
9.17.2 Constructor & Destructor Documentation . . . . .	69
9.17.2.1 <a href="#">HyRect()</a> . . . . .	69
9.17.3 Member Function Documentation . . . . .	70
9.17.3.1 <a href="#">divide()</a> . . . . .	70
9.17.3.2 <a href="#">getAvgValue()</a> . . . . .	70
9.17.3.3 <a href="#">getD()</a> . . . . .	70
9.17.3.4 <a href="#">getDepth()</a> . . . . .	71
9.17.3.5 <a href="#">getDiagonalLength()</a> . . . . .	71
9.17.3.6 <a href="#">getPos()</a> . . . . .	71
9.17.3.7 <a href="#">getSamplingVertices()</a> . . . . .	71
9.17.3.8 <a href="#">getSplitDim()</a> . . . . .	71
9.17.3.9 <a href="#">operator!=(())</a> . . . . .	72
9.17.3.10 <a href="#">operator&lt;()</a> . . . . .	72
9.17.3.11 <a href="#">operator&lt;=()</a> . . . . .	72
9.17.3.12 <a href="#">operator==(())</a> . . . . .	73
9.17.3.13 <a href="#">operator&gt;()</a> . . . . .	73
9.17.3.14 <a href="#">operator&gt;=()</a> . . . . .	73
9.17.3.15 <a href="#">setAvgValue()</a> . . . . .	74
9.17.4 Member Data Documentation . . . . .	74
9.17.4.1 <a href="#">avgValue</a> . . . . .	74
9.17.4.2 <a href="#">D</a> . . . . .	74
9.17.4.3 <a href="#">pos</a> . . . . .	74
9.17.4.4 <a href="#">t</a> . . . . .	74
9.18 <a href="#">Levels Class Reference</a> . . . . .	75
9.18.1 Detailed Description . . . . .	76
9.18.2 Member Function Documentation . . . . .	76
9.18.2.1 <a href="#">getEpsilon()</a> . . . . .	76
9.18.2.2 <a href="#">getLevel()</a> . . . . .	76
9.18.2.3 <a href="#">getRectSubset()</a> . . . . .	76
9.18.2.4 <a href="#">isGlobal()</a> . . . . .	77
9.18.2.5 <a href="#">nextLevel()</a> . . . . .	77
9.18.2.6 <a href="#">setGlobal()</a> . . . . .	77



9.18.3 Member Data Documentation	77
9.18.3.1 currentLevel	77
9.18.3.2 global	78
9.18.3.3 L0_EPSILON	78
9.18.3.4 L0_SIZE	78
9.18.3.5 L1_EPSILON	78
9.18.3.6 L1_SIZE	78
9.18.3.7 L2_EPSILON	78
9.18.3.8 L2_SIZE	78
9.18.3.9 L3_EPSILON	78
9.18.3.10 L3_SIZE	79
9.19 Multithreaded< Key, T, Compare, Allocator > Class Template Reference	79
9.19.1 Detailed Description	79
9.19.2 Constructor & Destructor Documentation	80
9.19.2.1 Multithreaded()	80
9.19.3 Member Function Documentation	80
9.19.3.1 multithreadFunction()	80
9.19.3.2 runMultithreadedFunctions()	80
9.19.3.3 work()	81
9.19.4 Member Data Documentation	81
9.19.4.1 NR_THREADS	81
9.19.4.2 queue	81
9.20 Optimizer Class Reference	82
9.20.1 Detailed Description	83
9.20.2 Constructor & Destructor Documentation	83
9.20.2.1 Optimizer()	83
9.20.3 Member Function Documentation	83
9.20.3.1 getName()	84
9.20.3.2 getStatus()	84
9.20.3.3 getStatusBar()	84
9.20.3.4 getValueMap()	84
9.20.3.5 requestValues()	85
9.20.3.6 runOptimization()	86
9.20.4 Member Data Documentation	86
9.20.4.1 controller	86
9.20.4.2 parameters	86
9.21 Parameter Class Reference	86
9.21.1 Detailed Description	87
9.21.2 Constructor & Destructor Documentation	88
9.21.2.1 Parameter()	88
9.21.3 Member Function Documentation	88
9.21.3.1 getConfig()	88

9.21.3.2 getMax()	88
9.21.3.3 getMin()	88
9.21.3.4 getUnit()	89
9.21.3.5 getVal()	89
9.21.3.6 operator"!="()	89
9.21.3.7 operator<()	89
9.21.3.8 operator<=()	90
9.21.3.9 operator==()	90
9.21.3.10 operator>()	90
9.21.3.11 operator>=()	91
9.21.3.12 setVal()	91
9.21.4 Member Data Documentation	91
9.21.4.1 definition	91
9.22 ParameterDefinition Class Reference	91
9.22.1 Detailed Description	92
9.22.2 Constructor & Destructor Documentation	92
9.22.2.1 ParameterDefinition()	92
9.22.3 Member Function Documentation	92
9.22.3.1 getConfig()	93
9.22.3.2 getMax()	93
9.22.3.3 getMin()	93
9.22.3.4 getUnit()	93
9.22.4 Member Data Documentation	93
9.22.4.1 config	93
9.22.4.2 max	94
9.22.4.3 min	94
9.22.4.4 unit	94
9.23 ParameterNormalizer Class Reference	94
9.23.1 Detailed Description	94
9.23.2 Constructor & Destructor Documentation	95
9.23.2.1 ParameterNormalizer()	95
9.23.3 Member Function Documentation	95
9.23.3.1 denormalize()	95
9.23.3.2 normalize()	95
9.23.4 Member Data Documentation	95
9.23.4.1 parameters	96
9.24 PlexeSimulationRunner Class Reference	96
9.24.1 Detailed Description	97
9.24.2 Constructor & Destructor Documentation	97
9.24.2.1 PlexeSimulationRunner()	97
9.24.3 Member Function Documentation	98
9.24.3.1 getName()	98

9.24.3.2 getRunId()	98
9.24.3.3 getStatus()	98
9.24.3.4 getStatusBar()	98
9.24.3.5 work() [1/2]	99
9.24.3.6 work() [2/2]	99
9.24.4 Member Data Documentation	99
9.24.4.1 editor	99
9.24.4.2 REPEAT	99
9.24.4.3 runNumber	100
9.24.4.4 runNumberLock	100
9.24.4.5 SCENARIOS	100
9.25 PythonScript Class Reference	100
9.25.1 Detailed Description	101
9.25.2 Constructor & Destructor Documentation	101
9.25.2.1 PythonScript()	101
9.25.2.2 ~PythonScript()	101
9.25.3 Member Data Documentation	101
9.25.3.1 pFunc	101
9.25.3.2 pModule	101
9.26 SimulationRunner Class Reference	102
9.26.1 Detailed Description	103
9.26.2 Constructor & Destructor Documentation	103
9.26.2.1 SimulationRunner()	103
9.26.3 Member Function Documentation	103
9.26.3.1 getName()	103
9.26.3.2 getStatus()	103
9.26.3.3 getStatusBar()	104
9.26.3.4 runSimulations()	104
9.26.3.5 work()	104
9.27 Status Class Reference	105
9.27.1 Detailed Description	105
9.27.2 Member Function Documentation	105
9.27.2.1 getName()	105
9.27.2.2 getStatus()	106
9.27.2.3 getStatusBar()	106
9.27.3 Member Data Documentation	106
9.27.3.1 NO_NAME	106
9.27.3.2 NO_STATUS_SUPPORT	107
9.28 StatusBar Class Reference	107
9.28.1 Detailed Description	108
9.28.2 Member Function Documentation	108
9.28.2.1 printResult()	108

9.28.2.2 printResults()	108
9.28.2.3 printStatus()	108
9.28.2.4 updateStatus()	109
9.28.3 Member Data Documentation	109
9.28.3.1 LARGE_DIVIDER	109
9.28.3.2 lastStatus	109
9.28.3.3 lastStep	109
9.28.3.4 lastVal	110
9.28.3.5 SMALL_DIVIDER	110
9.29 Controller::stepstate Struct Reference	110
9.29.1 Detailed Description	110
9.29.2 Member Function Documentation	110
9.29.2.1 get()	110
9.29.2.2 next()	111
9.29.3 Member Data Documentation	111
9.29.3.1 currentStep	111
9.29.3.2 stepChanged	111
9.30 StoppingCondition Class Reference	111
9.30.1 Detailed Description	112
9.30.2 Constructor & Destructor Documentation	112
9.30.2.1 StoppingCondition() [1/2]	112
9.30.2.2 StoppingCondition() [2/2]	113
9.30.3 Member Function Documentation	113
9.30.3.1 evaluate()	113
9.30.3.2 getIterationsSinceImprov()	113
9.30.3.3 setStartNow()	114
9.30.3.4 updateAccuracy()	114
9.30.4 Member Data Documentation	114
9.30.4.1 ACCURACY	114
9.30.4.2 bestVal	114
9.30.4.3 END_TIME	114
9.30.4.4 iterationsSinceImprov	115
9.30.4.5 mins	115
9.30.4.6 NR_ACCURACY_ITERATIONS	115
9.30.4.7 NR_EVALUATIONS	115
9.30.4.8 NR_HYRECTS	115
9.30.4.9 time_eval	115
9.31 StubController Class Reference	115
9.31.1 Detailed Description	117
9.31.2 Constructor & Destructor Documentation	117
9.31.2.1 StubController()	117
9.31.3 Member Function Documentation	117

9.31.3.1 evaluate()	117
9.31.3.2 removeOldResultfiles()	118
9.31.3.3 runSimulations()	118
9.31.3.4 updateStatus()	118
9.31.4 Member Data Documentation	118
9.31.4.1 f	119
9.31.4.2 functions	119
9.32 ThreadsafeQueue< Key > Class Template Reference	119
9.32.1 Detailed Description	119
9.32.2 Member Function Documentation	120
9.32.2.1 getSize()	120
9.32.2.2 getStartSize()	120
9.32.2.3 pop()	120
9.32.2.4 push()	120
9.32.3 Member Data Documentation	120
9.32.3.1 queueLock	121
9.32.3.2 safeQueue	121
9.32.3.3 startSize	121
9.33 ValueMap Class Reference	121
9.33.1 Detailed Description	122
9.33.2 Constructor & Destructor Documentation	122
9.33.2.1 ValueMap()	122
9.33.3 Member Function Documentation	122
9.33.3.1 addValue()	123
9.33.3.2 getMedian()	123
9.33.3.3 getSize()	123
9.33.3.4 getTopVals()	123
9.33.3.5 getValues()	124
9.33.3.6 insert()	124
9.33.3.7 isKnown()	124
9.33.3.8 isTopValue()	124
9.33.3.9 query()	125
9.33.3.10 updateMap()	125
9.33.4 Member Data Documentation	125
9.33.4.1 lowerValues	125
9.33.4.2 operationsLock	125
9.33.4.3 tba	125
9.33.4.4 topEntries	126
9.33.4.5 topVals	126
9.33.4.6 upperValues	126
9.33.4.7 values	126

<b>10 File Documentation</b>	<b>127</b>
10.1 /home/runner/work/simopticon/simopticon/src/ComparisonFunctions.h File Reference	127
10.1.1 Detailed Description	128
10.2 /home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/constant_headway.py File Reference	128
10.2.1 Detailed Description	128
10.2.2 Function Documentation	128
10.2.2.1 get_constant_headway()	129
10.2.2.2 get_last_value()	129
10.2.2.3 multithreaded()	129
10.3 /home/runner/work/simopticon/simopticon/src/main.cpp File Reference	130
10.3.1 Detailed Description	130
10.3.2 Function Documentation	130
10.3.2.1 interruptHandler()	130
10.3.2.2 main()	131
10.3.3 Variable Documentation	131
10.3.3.1 ctr	131
10.4 /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectComparisonFunctions.h File Reference	131
10.4.1 Detailed Description	132
10.5 /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectTypes.h File Reference	132
10.5.1 Detailed Description	133
10.5.2 Typedef Documentation	133
10.5.2.1 depth	133
10.5.2.2 dimension	133
10.5.2.3 dirCoordinate	133
10.6 /home/runner/work/simopticon/simopticon/src/Types.h File Reference	133
10.6.1 Detailed Description	134
10.6.2 Typedef Documentation	134
10.6.2.1 coordinate	134
10.6.2.2 functionValue	134
10.6.2.3 runId	135
<b>Index</b>	<b>137</b>

# Chapter 1

## Documentation

- 
1. [Overview](#)
  2. [Setup](#)
    - (a) [Requirements](#)
    - (b) [Installation](#)
  3. [Usage](#)
    - (a) [Configuration](#)
    - (b) [Optimization](#)
  4. [Extension](#)
    - (a) [Development](#)
    - (b) [Integration](#)
- 

### 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](https://peternaggschga.github.io/simopticon) or in the comprehensive [PDF file](#) provided.

---

## 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. `pythonScript` 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.

Please document your code using [Doxygen](#) comments!

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.

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:

controller . . . . .	19
parameters . . . . .	22
evaluation . . . . .	22
constant_headway . . . . .	21
runner . . . . .	23
plexo . . . . .	21
status . . . . .	23
optimizer . . . . .	24
direct . . . . .	19
hyrect . . . . .	24
utils . . . . .	25



## Chapter 5

# Hierarchical Index

### 5.1 Class Hierarchy

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

Abortable	27
Controller	48
StubController	115
Optimizer	82
DirectOptimizer	54
CmpPairVectorSharedParameterFunctionvalue	32
CmpPtrFunctionvalue	33
CmpSharedHyrect	33
CmpVectorSharedParameter	34
CommandLine	35
ConfigEditor	36
GrahamScan	67
HyRect	68
BaseRect	28
ChildRect	30
Levels	75
Multithreaded< Key, T, Compare, Allocator >	79
Multithreaded< pair< filesystem::path, pair< string, unsigned int > >, bool >	79
PlexeSimulationRunner	96
Multithreaded< vector< shared_ptr< Parameter > >, pair< filesystem::path, set< runId > >, Cmp← VectorSharedParameter >	79
SimulationRunner	102
PlexeSimulationRunner	96
Parameter	86
ContinuousParameter	44
DiscreteParameter	60
ParameterDefinition	91
ParameterNormalizer	94
PythonScript	100
ConstantHeadway	40
Status	105
Evaluation	65
ConstantHeadway	40
Optimizer	82
SimulationRunner	102
StatusBar	107
Controller::stepstate	110
StoppingCondition	111

ThreadsafeQueue< Key > . . . . .	<a href="#">119</a>
ValueMap . . . . .	<a href="#">121</a>

## Chapter 6

# Class Index

### 6.1 Class List

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

<a href="#">Abortable</a>	A simple interface for classes that encapsulate abortable processes . . . . .	27
<a href="#">BaseRect</a>	A class representing a <a href="#">HyRect</a> without a parent rectangle . . . . .	28
<a href="#">ChildRect</a>	A class representing a <a href="#">HyRect</a> that has a parent <a href="#">HyRect</a> . . . . .	30
<a href="#">CmpPairVectorSharedParameterFunctionvalue</a>	This struct implements the comparison of two pairs of <a href="#">Parameter</a> combination and function value . . . . .	32
<a href="#">CmpPtrFunctionvalue</a>	This struct implements the comparison of two pointers to function values . . . . .	33
<a href="#">CmpSharedHyrect</a>	This struct implements the comparison of two shared pointers to <a href="#">HyRect</a> instances . . . . .	33
<a href="#">CmpVectorSharedParameter</a>	This struct implements the comparison of two vectors of <a href="#">Parameter</a> references . . . . .	34
<a href="#">CommandLine</a>	A class containing functionality for executing commands on UNIX shell . . . . .	35
<a href="#">ConfigEditor</a>	A class capable of creating <code>.ini</code> files with certain options based on a complete <code>omnetpp.ini</code> . . . . .	36
<a href="#">ConstantHeadway</a>	A wrapper for the <code>constant_headway.py</code> script . . . . .	40
<a href="#">ContinuousParameter</a>	Implements a <a href="#">Parameter</a> using continuous values in the form of floating point numbers . . . . .	44
<a href="#">Controller</a>	A class responsible for communication between <a href="#">Optimizer</a> , <a href="#">SimulationRunner</a> and <a href="#">Evaluation</a> and also user interaction such as tracking results, updating <a href="#">StatusBar</a> and handling interrupts by the user via <a href="#">Abortable</a> . . . . .	48
<a href="#">DirectOptimizer</a>	A class capable of finding the minimum of a blackbox function using the DIRECT algorithm . . . . .	54
<a href="#">DiscreteParameter</a>	Implements a <a href="#">Parameter</a> using discrete values . . . . .	60
<a href="#">Evaluation</a>	A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization . . . . .	65
<a href="#">GrahamScan</a>	A class providing functionality for finding the lower right convex hull of a set of points . . . . .	67
<a href="#">HyRect</a>	An abstract class representing a rectangular part of the search space . . . . .	68
<a href="#">Levels</a>	A providing functionality for the usage of different weightings between local and global search throughout the optimization using different levels . . . . .	75

<a href="#">Multithreaded&lt; Key, T, Compare, Allocator &gt;</a>	
A class implementing concurrent execution of the same function for different arguments . . . .	79
<a href="#">Optimizer</a>	
A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs . . . . .	82
<a href="#">Parameter</a>	
A class acting as the container of the value of a parameter defined by a <a href="#">ParameterDefinition</a> .	86
<a href="#">ParameterDefinition</a>	
A class storing information on the properties of parameters that are being optimized . . . . .	91
<a href="#">ParameterNormalizer</a>	
A class used for transforming parameters between the actual <a href="#">Parameter</a> space and the unit hypercube used in DIRECT algorithm . . . . .	94
<a href="#">PlexeSimulationRunner</a>	
A class capable of starting platooning simulations in the <a href="#">Plexe</a> framework with given <a href="#">Parameter</a> combinations . . . . .	96
<a href="#">PythonScript</a>	
A class containing functionality for interfacing with the function of a Python module on creation	100
<a href="#">SimulationRunner</a>	
A class capable of running simulations with certain <a href="#">Parameter</a> combinations . . . . .	102
<a href="#">Status</a>	
An interface defining functions for status updates on configuration and progress of a class . . .	105
<a href="#">StatusBar</a>	
A class used to conduct command line output containing information about the state of the used <a href="#">Optimizer</a> , <a href="#">SimulationRunner</a> and <a href="#">Evaluation</a> along with the found optima . . . . .	107
<a href="#">Controller::stepstate</a>	
A struct keeping track of the currently running optimization step for <a href="#">StatusBar::updateStatus</a> . .	110
<a href="#">StoppingCondition</a>	
A class used for deciding whether the DIRECT should be stopped . . . . .	111
<a href="#">StubController</a>	
A class that mocks behaviour of <a href="#">Controller</a> . . . . .	115
<a href="#">ThreadsafeQueue&lt; Key &gt;</a>	
A container class of a queue that is safe for concurrent access of different threads . . . . .	119
<a href="#">ValueMap</a>	
A container managing a map data structure that maps <a href="#">Parameter</a> combinations to their respective found values . . . . .	121

## Chapter 7

# File Index

### 7.1 File List

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

/home/runner/work/simopticon/simopticon/src/ <a href="#">ComparisonFunctions.h</a>	
In this file comparison functions are defined which should be used across the whole framework	127
/home/runner/work/simopticon/simopticon/src/ <a href="#">main.cpp</a>	
Definition of the main function running the <i>Simopticon</i> framework	130
/home/runner/work/simopticon/simopticon/src/ <a href="#">Types.h</a>	
In this file types are defined which should be used across the whole framework	133
/home/runner/work/simopticon/simopticon/src/controller/ <b>Controller.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/controller/ <b>Controller.h</b>	??
/home/runner/work/simopticon/simopticon/src/controller/ <b>StubController.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/controller/ <b>StubController.h</b>	??
/home/runner/work/simopticon/simopticon/src/controller/ <b>ValueMap.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/controller/ <b>ValueMap.h</b>	??
/home/runner/work/simopticon/simopticon/src/evaluation/ <b>Evaluation.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/evaluation/ <b>Evaluation.h</b>	??
/home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/ <a href="#">constant_headway.py</a>	
A Python script providing functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap	128
/home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/ <b>ConstantHeadway.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/evaluation/constant_headway/ <b>ConstantHeadway.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/ <b>Optimizer.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/ <b>Optimizer.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <a href="#">DirectComparisonFunctions.h</a>	
In this file comparison functions are defined which are used in the direct module	131
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>DirectOptimizer.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>DirectOptimizer.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <a href="#">DirectTypes.h</a>	
In this file types are defined which are in the direct module	132
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>GrahamScan.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>GrahamScan.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>Levels.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>Levels.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>ParameterNormalizer.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>ParameterNormalizer.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>StoppingCondition.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/ <b>StoppingCondition.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ <b>BaseRect.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ <b>BaseRect.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ <b>ChildRect.cpp</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ <b>ChildRect.h</b>	??
/home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ <b>HyRect.cpp</b>	??

/home/runner/work/simopticon/simopticon/src/optimizer/direct/hyrect/ <b>HyRect.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>ContinuousParameter.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>ContinuousParameter.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>DiscreteParameter.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>DiscreteParameter.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>Parameter.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>Parameter.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>ParameterDefinition.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/parameters/ <b>ParameterDefinition.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/runner/ <b>SimulationRunner.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/runner/ <b>SimulationRunner.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/runner/plex/ <b>ConfigEditor.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/runner/plex/ <b>ConfigEditor.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/runner/plex/ <b>PlexeSimulationRunner.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/runner/plex/ <b>PlexeSimulationRunner.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/status/ <b>Status.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/status/ <b>Status.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/status/ <b>StatusBar.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/status/ <b>StatusBar.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>Abortable.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>Abortable.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>CommandLine.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>CommandLine.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>Multithreaded.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>Multithreaded.hpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>PythonScript.cpp</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>PythonScript.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>ThreadsafeQueue.h</b> . . . . .	??
/home/runner/work/simopticon/simopticon/src/utils/ <b>ThreadsafeQueue.hpp</b> . . . . .	??



## 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 [StubController](#)  
*A class that mocks behaviour of [Controller](#).*
- class [ValueMap](#)  
*A container managing a map data structure that maps [Parameter](#) combinations to their respective found values.*

#### 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

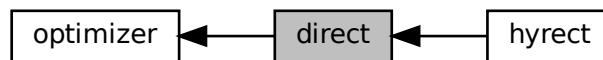
```
struct Controller::stepstate Controller::stepState [protected]
```

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 {  
`I2_0 = 0 , I1_1 = 1 , I0_2 = 2 , I1_3 = 3 ,  
I1_4 = 4 , I0_5 = 5 , I1_6 = 6 , I2_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 pre-defined 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 [ContinuousParameter](#)  
Implements a [Parameter](#) using continuous values in the form of floating point numbers.
- class [ParameterDefinition](#)  
A class storing information on the properties of parameters that are being optimized.
- class [DiscreteParameter](#)  
Implements a [Parameter](#) using discrete values.

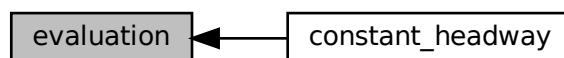
### 8.5.1 Detailed Description

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 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.*

### 8.7.1 Detailed Description

This module contains components capable of automatically running simulations with certain [Parameter](#) combinations.

Implementations must extend [SimulationRunner](#).

## 8.8 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.*

### 8.8.1 Detailed Description

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

### 8.8.2 Enumeration Type Documentation

#### 8.8.2.1 step

```
enum step : char
```

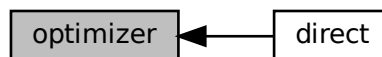
An Enum defining the steps, an optimization process cycles through.

Definition at line 27 of file StatusBar.h.

## 8.9 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.*

### 8.9.1 Detailed Description

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

Implementations must extend [Optimizer](#).

## 8.10 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.*
- 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/utls/Abortable.h](#)
- [/home/runner/work/simopticon/simopticon/src/utls/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

<i>pos</i>	Relative position to the given parent rectangle.
<i>parent</i>	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

<i>rect</i>	<a href="#">HyRect</a> 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

<i>a</i>	First pair.
<i>b</i>	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

<i>a</i>	First pointer to a function value.
<i>b</i>	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

<i>a</i>	First pointer to a <a href="#">HyRect</a> .
<i>b</i>	Second pointer to a <a href="#">HyRect</a> .

#### 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

<i>a</i>	First vector to be compared.
<i>b</i>	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

<i>cmd</i>	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

<i>directory</i>	A path to the directory containing a Plexe configuration.
<i>controller</i>	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

<i>params</i>	The <a href="#">Parameter</a> combination to be simulated.
<i>runNumber</i>	An unique number of the simulated <a href="#">Parameter</a> combination.
<i>repeat</i>	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

<i>runId</i>	Number of the configuration file to be deleted.
--------------	---

Definition at line 90 of file ConfigEditor.cpp.

References [getConfigPath\(\)](#).

### 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

<i>runId</i>	Number of the <a href="#">Parameter</a> combination.
--------------	--

#### Returns

A path to the config for the given runId.

Definition at line 80 of file ConfigEditor.cpp.

Referenced by deleteConfig().

### 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

<i>file</i>	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

<i>runId</i>	Number of the <a href="#">Parameter</a> 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**

<i>file</i>	A string containing the contents of an .ini file.
<i>option</i>	A string representing a key in the given file.
<i>value</i>	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**

<i>file</i>	A string containing the contents of an .ini file.
<i>option</i>	A string representing a key in the given file.
<i>value</i>	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**

<i>file</i>	A string containing the contents of an .ini file.
<i>runNumber</i>	The unique number of the <a href="#">Parameter</a> 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/plex/ConfigEditor.h`
- `/home/runner/work/simopticon/simopticon/src/runner/plex/ConfigEditor.cpp`

## 9.10 ConstantHeadway Class Reference

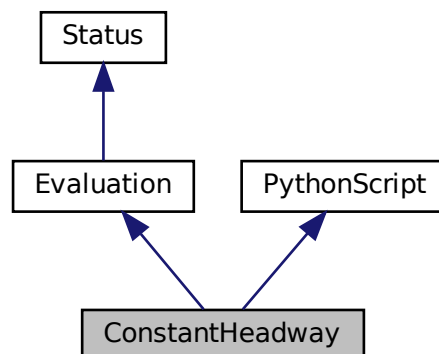
A wrapper for the [constant\\_headway.py](#) script.

```
#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

<i>nrThreads</i>	Maximum number of threads used for concurrent calculations.
<i>pathToScript</i>	Path to the <a href="#">constant_headway.py</a> script.

Definition at line 8 of file ConstantHeadway.cpp.

References `PythonScript::PythonScript()`, and `NR_THREADS`.

### 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

<i>experimentResults</i>	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.

References usedThreads.

### 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**

<i>path</i>	Path to the result files.
<i>experimentIds</i>	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**

<i>object</i>	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\(\)](#), and [processOutput\(\)](#).

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

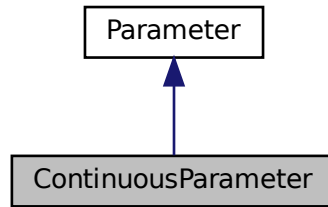
- [/home/runner/work/simopticon/simopticon/src/evaluation/constant\\_headway/ConstantHeadway.h](#)
- [/home/runner/work/simopticon/simopticon/src/evaluation/constant\\_headway/ConstantHeadway.cpp](#)

**9.11 ContinuousParameter Class Reference**

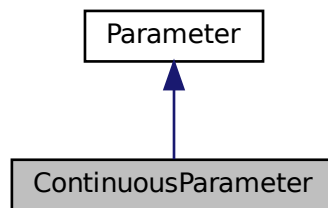
Implements a [Parameter](#) using continuous values in the form of floating point numbers.

```
#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 continuous 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

<i>def</i>	<a href="#">ParameterDefinition</a> of the <a href="#">Parameter</a> .
<i>value</i>	Initial value of the <a href="#">Parameter</a> .

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

<i>def</i>	<a href="#">ParameterDefinition</a> of the <a href="#">Parameter</a> .
------------	--

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

<code>val</code>	Value to set the <a href="#">ContinuousParameter</a> 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, bool isStub=false)  
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.
- struct [Controller::stepstate](#) [stepState](#)  
An object keeping track of the current optimization step.

## 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.*
- `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,
    bool isStub = false ) [explicit]
```

Creates a [Controller](#) which uses [Optimizer](#), [SimulationRunner](#) and [Evaluation](#) as specified in the given config files. If called by the constructor of [StubController](#), `runner` and `evaluation` get assigned null-pointers.

#### Parameters

<code>configPath</code>	Path to the main config. Chosen by first command line argument.
<code>isStub</code>	Defines whether the constructor was called by constructor of <a href="#">StubController</a> .

Definition at line 28 of file `Controller.cpp`.  
 References `statusInterval`, and `valueMap`.  
 Referenced by `StubController::StubController()`.

### 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 226 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

<i>simulationResults</i>	A map which maps the <a href="#">Parameter</a> 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 184 of file Controller.cpp.

References [updateStatus\(\)](#).

### 9.12.3.3 getValueMap()

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

Returns [valueMap](#).

#### Returns

A [ValueMap](#) object.

Definition at line 152 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 200 of file Controller.cpp.

References [ValueMap::getTopVals\(\)](#), [keepFiles](#), [topResults](#), and [valueMap](#).

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

<i>params</i>	A set of <a href="#">Parameter</a> combinations to be evaluated.
---------------	--

#### Returns

A map which maps the given [Parameter](#) combinations to their respective functionValue.

Definition at line 118 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 156 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

<i>runs</i>	A set of <a href="#">Parameter</a> 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 177 of file Controller.cpp.

#### 9.12.3.8 updateStatus()

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

Updates the [statusBar](#) using [StatusBar::updateStatus](#).

Reimplemented in [StubController](#).

Definition at line 218 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 97 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.

Referenced by `removeOldResultfiles()`.

#### 9.12.4.3 optimizer

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

[Optimizer](#) defining an optimization strategy.

Definition at line 89 of file Controller.h.

Referenced by `abort()`, `run()`, and `StubController::updateStatus()`.

#### 9.12.4.4 runner

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

[SimulationRunner](#) able to run simulations with certain [Parameter](#) combinations.

Definition at line 93 of file Controller.h.

#### 9.12.4.5 statusBar

`StatusBar Controller::statusBar [protected]`

[StatusBar](#) object used for output.

Definition at line 85 of file Controller.h.

Referenced by `StubController::updateStatus()`.

#### 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 50 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.

Referenced by `removeOldResultfiles()`.

#### 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 101 of file Controller.h.

Referenced by Controller(), getValueMap(), removeOldResultfiles(), and StubController::updateStatus().

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

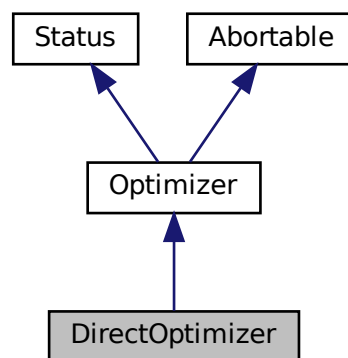
- /home/runner/work/simopticon/simopticon/src/controller/Controller.h
- /home/runner/work/simopticon/simopticon/src/controller/Controller.cpp

## 9.13 DirectOptimizer Class Reference

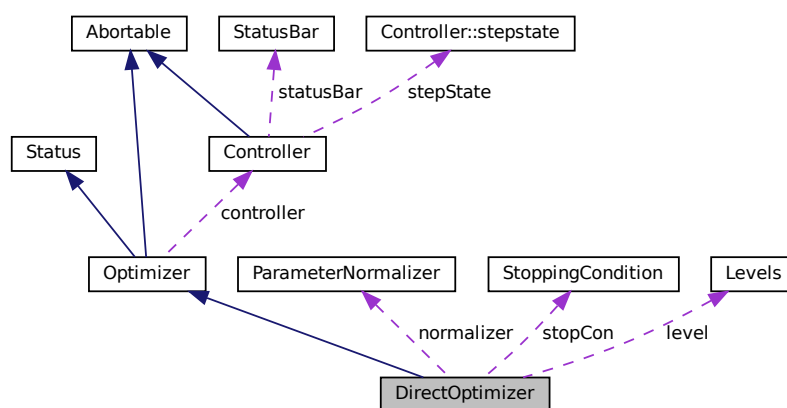
A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.

```
#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

<i>ctrl</i>	<a href="#">Controller</a> to be used for evaluating the optimized function.
<i>params</i>	<a href="#">ParameterDefinition</a> list to be optimized.
<i>con</i>	<a href="#">StoppingCondition</a> defining the end of optimization.
<i>trackProgress</i>	Defines whether the progress should be printed in a <code>.csv</code> file.
<i>printValues</i>	Defines whether all obtained values should be printed in a <code>.csv</code> 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

<i>rects</i>	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

<i>rect</i>	Rectangle the minimum is searched for.
<i>k</i>	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**

<i>points</i>	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**

<i>nrRects</i>	Size of the partition (meaning number of rectangles in <a href="#">activeRects</a> ).
<i>phi</i>	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**

<i>rects</i>	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

<i>bestVal</i>	Value at the current minimum.
<i>evaluations</i>	Number of evaluations conducted by the optimization.
<i>nrRects</i>	Number of rectangles in the current partition (meaning number of rectangles in <a href="#">activeRects</a> ).

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:

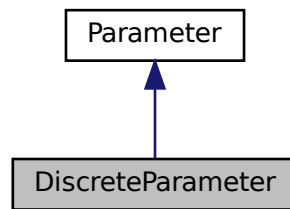
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectOptimizer.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectOptimizer.cpp

## 9.14 DiscreteParameter Class Reference

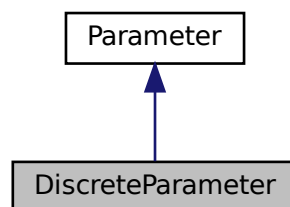
Implements a [Parameter](#) using discrete values.

```
#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

<i>def</i>	<a href="#">ParameterDefinition</a> of the <a href="#">Parameter</a> .
<i>step</i>	Difference between discrete values.
<i>value</i>	Initial value of the <a href="#">Parameter</a> .

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

<i>def</i>	<a href="#">ParameterDefinition</a> of the <a href="#">Parameter</a> .
<i>step</i>	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

<i>newTimes</i>	
-----------------	--

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

<i>val</i>	Value to set the <a href="#">DiscreteParameter</a> 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

<i>experimentResults</i>	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.

References [processOutput\(\)](#).

## 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

<i>path</i>	Path to the result files.
<i>experimentIds</i>	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](#).

Referenced by [processOutput\(\)](#).

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

<i>vertices</i>	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:

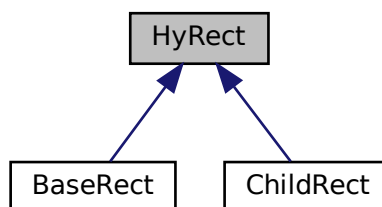
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/GrahamScan.h
- /home/runner/work/simopticon/simopticon/src/optimizer/direct/GrahamScan.cpp

## 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

<i>D</i>	Dimensionality of the rectangle (i.e. the search space).
<i>pos</i>	Position relative to parent rectangle.
<i>t</i>	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

<i>ptr</i>	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**

<i>rhs</i>	<a href="#">HyRect</a> 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**

<i>rect</i>	<a href="#">HyRect</a> 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**

<i>rhs</i>	<a href="#">HyRect</a> 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**

<code>rect</code>	<a href="#">HyRect</a> 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**

<code>rhs</code>	<a href="#">HyRect</a> 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**

<code>rhs</code>	<a href="#">HyRect</a> 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**

<code>value</code>	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

<i>rects</i>	Map containing all <a href="#">HyRect</a> of the current partition grouped by <a href="#">HyRect::t</a> and sorted by <a href="#">HyRect::avgValue</a> .
<i>size</i>	Number of <a href="#">HyRect</a> 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**

<i>val</i>	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 = 12_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

<a href="#">Key</a>	Argument type of the concurrent <a href="#">work</a> function.
---------------------	--

## Template Parameters

<i>T</i>	Result type of the concurrent <a href="#">work</a> function.
<i>Compare</i>	Comparison for objects of type Key.
<i>Allocator</i>	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 , class Allocator >
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

<i>threads</i>	Maximum number of threads to use.
----------------	-----------------------------------

Definition at line 6 of file Multithreaded.hpp.

References `Multithreaded< Key, T, Compare, Allocator >::NR_THREADS`.

Referenced by `SimulationRunner::SimulationRunner()`.

## 9.19.3 Member Function Documentation

### 9.19.3.1 multithreadFunction()

```
template<class Key , class T , class Compare , class Allocator >
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.

Definition at line 30 of file Multithreaded.hpp.

References `Multithreaded< Key, T, Compare, Allocator >::queue`, and `Multithreaded< Key, T, Compare, Allocator >::work()`.

Referenced by `Multithreaded< Key, T, Compare, Allocator >::runMultithreadedFunctions()`.

### 9.19.3.2 runMultithreadedFunctions()

```
template<class Key , class T , class Compare , class Allocator >
map< Key, T, Compare, Allocator > Multithreaded< Key, T, Compare, Allocator >::runMultithreaded↵
Functions (
    set< Key, Compare > runs ) [protected], [virtual]
```

Pushes given tasks into [queue](#), creates concurrent threads and merges them when execution is done.

## Parameters

<i>runs</i>	Set of arguments on which <a href="#">work</a> should to be executed.
-------------	---

## Returns

A map which maps arguments to their respective calculated values.

Definition at line 11 of file Multithreaded.hpp.

References `Multithreaded< Key, T, Compare, Allocator >::multithreadFunction()`, `Multithreaded< Key, T, Compare, Allocator >::NR_THREADS`, and `Multithreaded< Key, T, Compare, Allocator >::queue`.

Referenced by `SimulationRunner::runSimulations()`.

## 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

<i>arg</i>	Argument of the concurrently executed function.
------------	---

## Returns

Return value of the concurrently executed function.

Referenced by `Multithreaded< Key, T, Compare, Allocator >::multithreadFunction()`.

## 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.

Referenced by `Multithreaded< Key, T, Compare, Allocator >::Multithreaded()`, and `Multithreaded< Key, T, Compare, Allocator >::runMultithreadedFunctions()`.

## 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.

Referenced by `Multithreaded< Key, T, Compare, Allocator >::multithreadFunction()`, and `Multithreaded< Key, T, Compare, Allocator >::runMultithreadedFunctions()`.

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

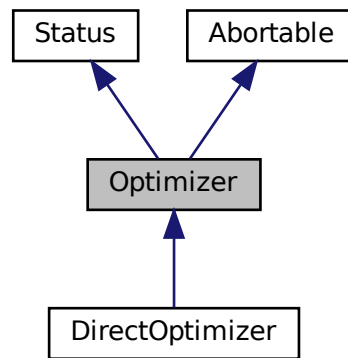
- `/home/runner/work/simopticon/simopticon/src/utils/Multithreaded.h`
- `/home/runner/work/simopticon/simopticon/src/utils/Multithreaded.hpp`

## 9.20 Optimizer Class Reference

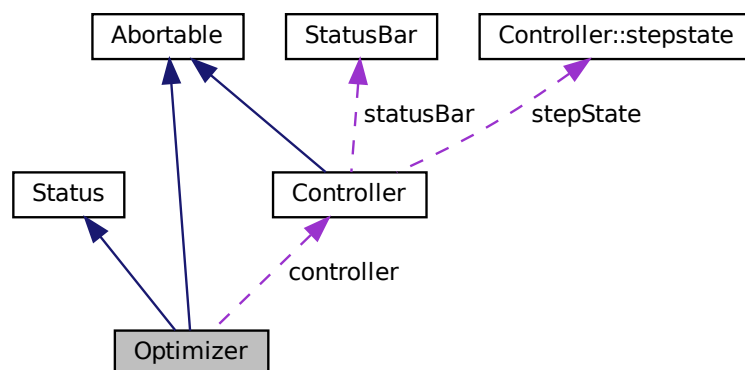
A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.

```
#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

<i>ctrl</i>	<i>Controller</i> to be used for evaluation of <i>Parameter</i> combinations.
<i>params</i>	List of <i>ParameterDefinition</i> 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

<i>params</i>	<a href="#">Parameter</a> 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:

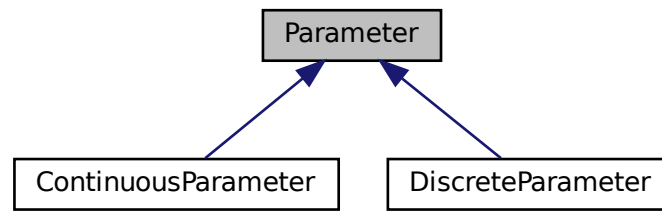
- `/home/runner/work/simopticon/simopticon/src/optimizer/Optimizer.h`
- `/home/runner/work/simopticon/simopticon/src/optimizer/Optimizer.cpp`

## 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

<i>def</i>	Definition of properties of the <a href="#">Parameter</a> .
------------	---

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

<i>rhs</i>	<a href="#">Parameter</a> 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

<i>rhs</i>	<a href="#">Parameter</a> 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**

<i>rhs</i>	<a href="#">Parameter</a> 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**

<i>rhs</i>	<a href="#">Parameter</a> 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**

<i>rhs</i>	<a href="#">Parameter</a> 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**

<i>rhs</i>	<a href="#">Parameter</a> 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**

<i>val</i>	Value to set the <a href="#">Parameter</a> 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

<i>min</i>	Minimum value of the <a href="#">Parameter</a> .
<i>max</i>	Maximum value of the <a href="#">Parameter</a> .
<i>config</i>	Configuration string for the <a href="#">Parameter</a> (optional).
<i>unit</i>	Unit of the <a href="#">Parameter</a> (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

<i>parameters</i>	<a href="#">ParameterDefinition</a> 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

<i>cords</i>	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

<i>params</i>	<a href="#">Parameter</a> 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:

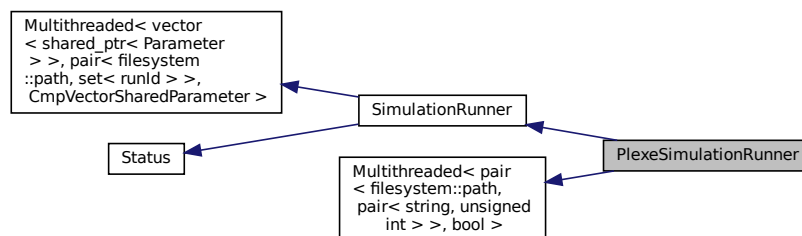
- `/home/runner/work/simopticon/simopticon/src/optimizer/direct/ParameterNormalizer.h`
- `/home/runner/work/simopticon/simopticon/src/optimizer/direct/ParameterNormalizer.cpp`

## 9.24 PlexeSimulationRunner Class Reference

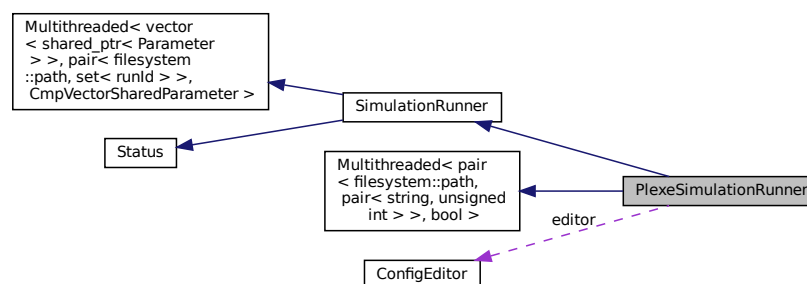
A class capable of starting platooning simulations in the [Plexe](#) framework with given [Parameter](#) combinations.

`#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

<i>threads</i>	Maximum number of threads to be used.
<i>repeat</i>	Number of repetitions per <a href="#">Parameter</a> combination and scenario.
<i>scenarios</i>	Scenarios to be simulated per <a href="#">Parameter</a> combination.
<i>editor</i>	<a href="#">ConfigEditor</a> to be used.

Definition at line 9 of file `PlexeSimulationRunner.cpp`.  
References [PlexeSimulationRunner\(\)](#), and [REPEAT](#).

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 77 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](#).

Referenced by [work\(\)](#).

#### 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 81 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 94 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

<i>arg</i>	A triple containing the path to the <code>.ini</code> defining the parameters, the scenario name and the repetition number.
------------	---

#### Returns

A boolean defining whether the execution ran without throwing exceptions.

Definition at line 62 of file `PlexeSimulationRunner.cpp`.

### 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

<i>run</i>	<a href="#">Parameter</a> 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`.

References `getRunId()`.

## 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 `PlexeSimulationRunner()`, and `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:

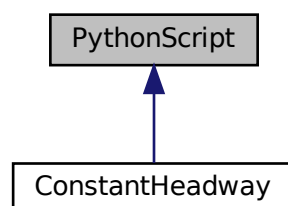
- /home/runner/work/simopticon/simopticon/src/runner/plex/PlexeSimulationRunner.h
- /home/runner/work/simopticon/simopticon/src/runner/plex/PlexeSimulationRunner.cpp

## 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

<i>path</i>	Path to the Python script containing the function.
<i>functionName</i>	Name of the function to be used.

Definition at line 8 of file PythonScript.cpp.  
Referenced by ConstantHeadway::ConstantHeadway().

#### 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:

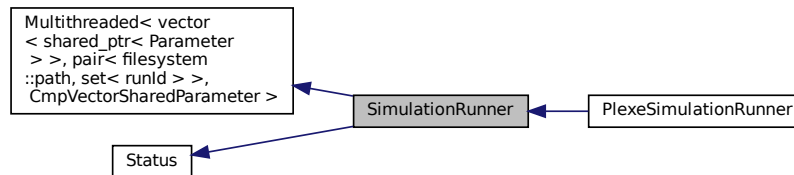
- /home/runner/work/simopticon/simopticon/src/utils/PythonScript.h
- /home/runner/work/simopticon/simopticon/src/utils/PythonScript.cpp

## 9.26 SimulationRunner Class Reference

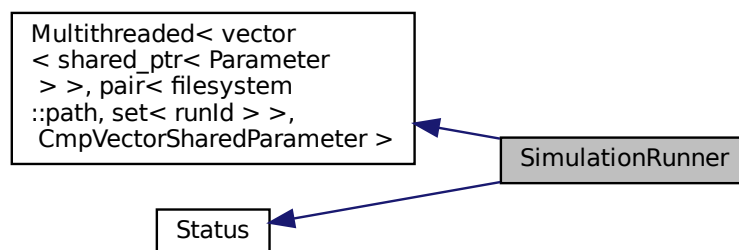
A class capable of running simulations with certain [Parameter](#) combinations.

```
#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

<i>threads</i>	Maximum number of threads that may be used for concurrent simulations.
----------------	--

Definition at line 6 of file SimulationRunner.cpp.  
References `Multithreaded< Key, T, Compare, Allocator >::Multithreaded()`.

### 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.  
References `Status::getName()`.

#### 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.  
References `Status::getStatus()`.

### 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`.

References `Status::getStatusBar()`.

### 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

<i>runs</i>	Set of <a href="#">Parameter</a> 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`.

References `Multithreaded< Key, T, Compare, Allocator >::runMultithreadedFunctions()`.

### 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

<i>run</i>	<a href="#">Parameter</a> 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:

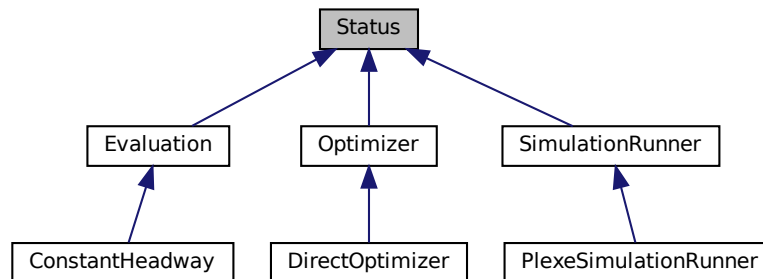
- `/home/runner/work/simopticon/simopticon/src/runner/SimulationRunner.h`
- `/home/runner/work/simopticon/simopticon/src/runner/SimulationRunner.cpp`

## 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()`, `SimulationRunner::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()`, `SimulationRunner::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()`, `SimulationRunner::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](#) ([Status](#) \*opt, [Status](#) \*runner, [Status](#) \*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

<i>cords</i>	<a href="#">Parameter</a> combination of the given result.
<i>optimum</i>	Value of the given result.

Definition at line 50 of file StatusBar.cpp.

References [Parameter::getConfig\(\)](#), [Parameter::getUnit\(\)](#), and [Parameter::getVal\(\)](#).

Referenced by [printResults\(\)](#), and [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

<i>top</i>	List of <a href="#">Parameter</a> combinations and respective values to be printed.
------------	---

Definition at line 69 of file StatusBar.cpp.

References [LARGE\\_DIVIDER](#), [printResult\(\)](#), and [SMALL\\_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

<i>object</i>	Object that inherits from <a href="#">Status</a> and whose state is being printed.
---------------	--

Definition at line 64 of file StatusBar.cpp.

References [Status::getName\(\)](#), and [Status::getStatus\(\)](#).

Referenced by [updateStatus\(\)](#).

### 9.28.2.4 updateStatus()

```
void StatusBar::updateStatus (
    Status * opt,
    Status * runner,
    Status * 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

<i>opt</i>	Pointer to <a href="#">Optimizer</a> used in optimization.
<i>runner</i>	Pointer to <a href="#">SimulationRunner</a> used in optimization.
<i>eval</i>	Pointer to <a href="#">Evaluation</a> used in optimization.
<i>currentVal</i>	<a href="#">Parameter</a> combination and respective value of the current optimum.
<i>stepChanged</i>	Boolean defining whether the current step has changed since the last call.
<i>currentStep</i>	Current step the optimization is in.

Definition at line 11 of file `StatusBar.cpp`.

References `Status::getStatusBar()`, `LARGE_DIVIDER`, `lastStatus`, `lastStep`, `lastVal`, `printResult()`, `printStatus()`, and `SMALL_DIVIDER`.

Referenced by `StubController::updateStatus()`.

## 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 [printResults\(\)](#), and [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](#).

```
#include "Controller.h"
```

### 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 107 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 129 of file Controller.h.

References `currentStep`, and `stepChanged`.

Referenced by `StubController::updateStatus()`.

#### 9.29.2.2 next()

```
void Controller::stepstate::next ( ) [inline]
```

Switches `currentStep` to the next step.

Definition at line 120 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 115 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 111 of file Controller.h.

Referenced by `get()`, `next()`, `Controller::updateStatus()`, and `StubController::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

<i>evaluations</i>	Number of evaluations after which the optimization should stop.
<i>hyrects</i>	Number of rectangles in the partition after which the optimization should stop.
<i>minutes</i>	Number of minutes after which the optimization should stop.
<i>accuracy</i>	Accuracy used in accuracy condition (see <a href="#">ACCURACY</a> ).
<i>accuracyIterations</i>	Number of iterations used in accuracy condition (see <a href="#">NR_ACCURACY_ITERATIONS</a> ).

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

<i>stopCon</i>	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

<i>evaluations</i>	Number of evaluations conducted by the optimization.
<i>hyrects</i>	Number of rectangles in the current partition.
<i>newBestVal</i>	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 `update_↵Accuracy()`.

### 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

<a href="#">newBestVal</a>	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:

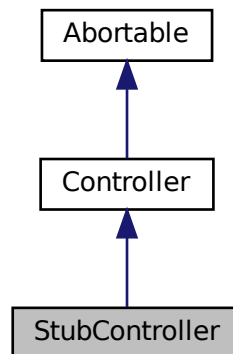
- `/home/runner/work/simopticon/simopticon/src/optimizer/direct/StoppingCondition.h`
- `/home/runner/work/simopticon/simopticon/src/optimizer/direct/StoppingCondition.cpp`

## 9.31 StubController Class Reference

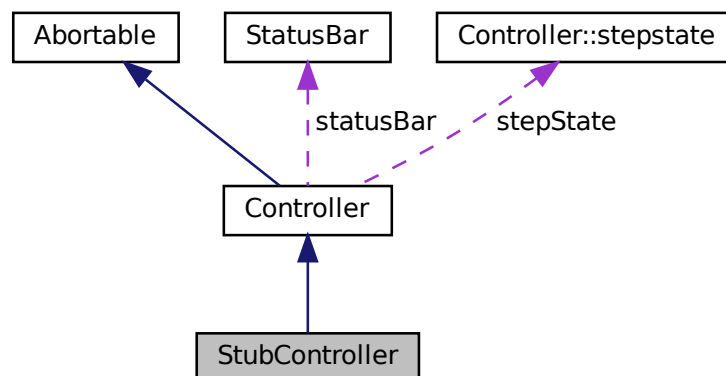
A class that mocks behaviour of [Controller](#).

```
#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.
- void [updateStatus](#) () override  
Updates the [statusBar](#) using [StatusBar::updateStatus](#).

## 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

<i>configPath</i>	Path to the main config. Chosen by first command line argument.
<i>function</i>	Name of the function to be used. Chosen by second command line argument.

Definition at line 119 of file [StubController.cpp](#).

References [Controller::Controller\(\)](#), *f*, and [functions](#).

### 9.31.3 Member Function Documentation

#### 9.31.3.1 evaluate()

```
map< vector< shared_ptr< Parameter > >, functionValue, CmpVectorSharedParameter > StubController::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

<i>simulationResults</i>	Map which maps <a href="#">Parameter</a> combinations to empty results (see <a href="#">runSimulations</a> ).
--------------------------	---

#### Returns

A Map which maps the given [Parameter](#) combinations to the respective value of [f](#).

Reimplemented from [Controller](#).

Definition at line 133 of file StubController.cpp.

References [f](#).

#### 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 142 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 runlds for each requested [Parameter](#) combination.

#### Parameters

<i>runs</i>	<a href="#">Parameter</a> combination to be simulated.
-------------	--

#### Returns

Map which maps the given [Parameter](#) combinations to empty paths and runlds.

Reimplemented from [Controller](#).

Definition at line 125 of file StubController.cpp.

#### 9.31.3.4 updateStatus()

```
void StubController::updateStatus ( ) [override], [private], [virtual]
```

Updates the [statusBar](#) using [StatusBar::updateStatus](#).

Reimplemented from [Controller](#).

Definition at line 145 of file StubController.cpp.

References [Controller::stepstate::get\(\)](#), [ValueMap::getSize\(\)](#), [ValueMap::getTopVals\(\)](#), [Controller::optimizer](#), [Controller::statusBar](#), [Controller::stepstate::stepChanged](#), [StatusBar::updateStatus\(\)](#), and [Controller::valueMap](#).

### 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(), and evaluate().

#### 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

<i>Key</i>	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

<i>val</i>	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/utls/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

<code>topEntries</code>	Value to be assigned to <code>topEntries</code> .
-------------------------	---

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**

<i>val</i>	<a href="#">Parameter</a> combination and respective value to be inserted.
<i>set</i>	Set that value is inserted in. Either <a href="#">lowerValues</a> or <a href="#">upperValues</a> .

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\(\)](#), [isKnown\(\)](#), and [StubController::updateStatus\(\)](#).

**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\(\)](#).

Referenced by [Controller::removeOldResultfiles\(\)](#), and [StubController::updateStatus\(\)](#).

### 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

<i>params</i>	<a href="#">Parameter</a> combination to be added.
<i>val</i>	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

<i>cords</i>	<a href="#">Parameter</a> 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

<i>cords</i>	<a href="#">Parameter</a> 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**

<i>params</i>	<a href="#">Parameter</a> 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.

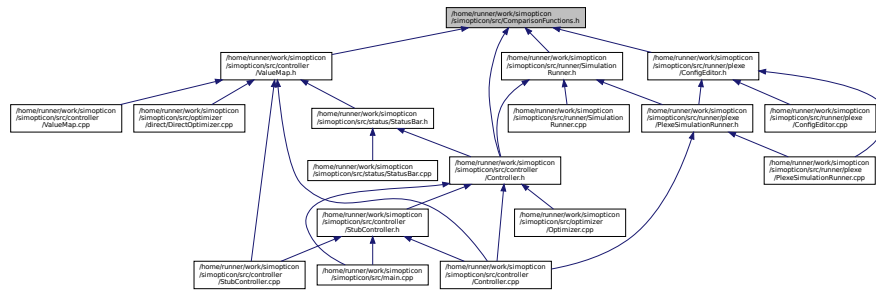
```
#include "parameters/Parameter.h"
```

```
#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

<i>run_ids</i>	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

<i>df</i>	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

<i>threads</i>	Maximum number of threads to be used for concurrent execution.
<i>directory</i>	A path to the directory directly or indirectly containing all result files that are to be evaluated.
<i>run_ids</i>	A list of lists of strings where each list of strings contains all OMNeT++ run ids of the runs conducted for one <a href="#">Parameter</a> combination

#### Returns

A list of longfloats representing the rating of the given simulation runs.

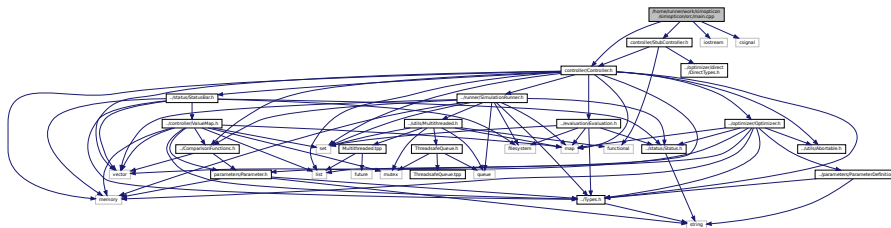
Definition at line 78 of file `constant_headway.py`.

## 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.



## 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.

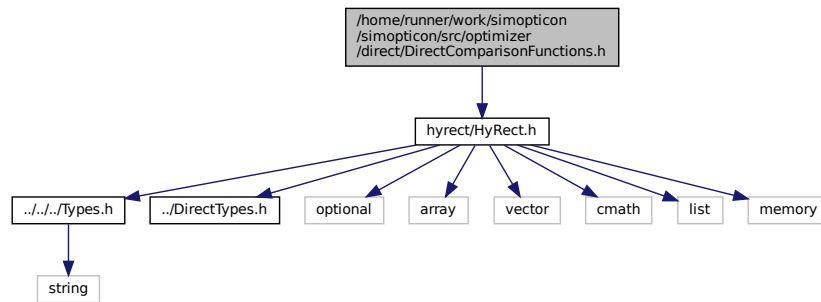
Definition at line 17 of file main.cpp.

## 10.4 /home/runner/work/simopticon/simopticon/src/optimizer/direct/↵ DirectComparisonFunctions.h File Reference

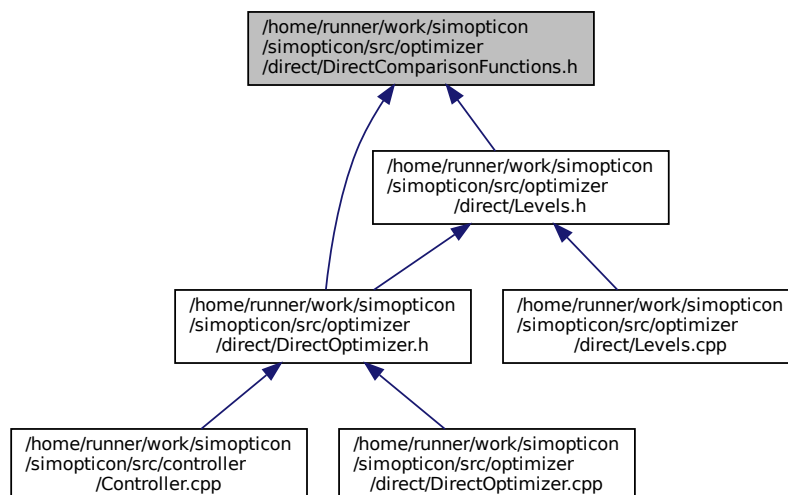
In this file comparison functions are defined which are used in the direct module.

```
#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.

- 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.*

In this file types are defined which are in the direct module.

### 10.5.2.1 depth

### 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.

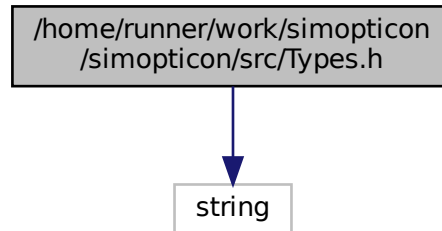
Definition at line 20 of file DirectTypes.h.

## 10.6 /home/runner/work/simopticon/simopticon/src/Types.h File Reference

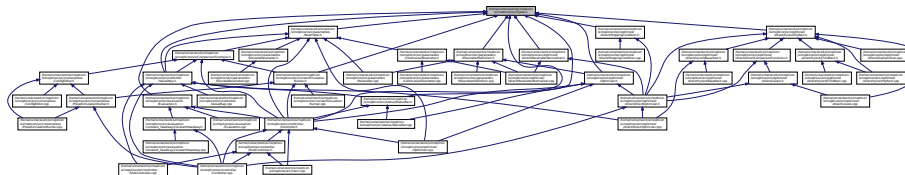
Generated by Doxygen

```
#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

[/home/runner/work/simopticon/simopticon/src/ComparisonFunction.h](#), 34  
[127](#)  
[/home/runner/work/simopticon/simopticon/src/Types.h](#), 133  
[/home/runner/work/simopticon/simopticon/src/evaluation/constant\\_headway/constant\\_headway.py](#), 128  
[/home/runner/work/simopticon/simopticon/src/main.cpp](#), 130  
[/home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectOptimizationFunctions.h](#), 131  
[/home/runner/work/simopticon/simopticon/src/optimizer/direct/DirectTypes.h](#), 132  
~PythonScript  
    PythonScript, [101](#)  
  
abort  
    Abortable, [28](#)  
    Controller, [50](#)  
Abortable, [27](#)  
    abort, [28](#)  
    aborted, [28](#)  
aborted  
    Abortable, [28](#)  
ACCURACY  
    StoppingCondition, [114](#)  
activeRects  
    DirectOptimizer, [59](#)  
addActiveRects  
    DirectOptimizer, [56](#)  
addValue  
    ValueMap, [122](#)  
avgValue  
    HyRect, [74](#)  
  
BaseRect, [28](#)  
    BaseRect, [29](#)  
    getSamplingVertices, [29](#)  
bestVal  
    StoppingCondition, [114](#)  
  
ChildRect, [30](#)  
    ChildRect, [31](#)  
    getSamplingVertices, [31](#)  
    operator==, [31](#)  
    parent, [32](#)  
CmpPairVectorSharedParameterFunctionvalue, [32](#)  
    operator(), [32](#)  
CmpPtrFunctionvalue, [33](#)  
    operator(), [33](#)  
CmpSharedHyrect, [33](#)  
    operator(), [34](#)  
CmpVectorSharedParameter, [34](#)  
    operator(), [34](#)  
CommandLine, [35](#)  
createConfig, [37](#)  
deleteConfig, [37](#)  
DIR, [40](#)  
getConfigPath, [37](#)  
getControllerOption, [38](#)  
getDir, [38](#)  
getResultPath, [38](#)  
replaceOption, [39](#)  
RESULTS, [40](#)  
setResultFiles, [39](#)  
constant\_headway, [21](#)  
constant\_headway.py  
    get\_constant\_headway, [128](#)  
    get\_last\_value, [129](#)  
    multithreaded, [129](#)  
ConstantHeadway, [40](#)  
    ConstantHeadway, [42](#)  
    getName, [42](#)  
    getStatus, [42](#)  
    getStatusBar, [43](#)  
    NR\_THREADS, [44](#)  
    processOutput, [43](#)  
    secureValue, [44](#)  
    usedThreads, [44](#)  
ContinuousParameter, [44](#)  
    ContinuousParameter, [46](#)  
    getVal, [46](#)  
    setVal, [46](#)  
    val, [48](#)  
CONTROLLER  
    ConfigEditor, [40](#)  
Controller, [48](#)  
    abort, [50](#)  
    Controller, [50](#)  
    evaluate, [51](#)  
    evaluation, [52](#)

- getValueMap, 51
  - keepFiles, 53
  - optimizer, 53
  - removeOldResultfiles, 51
  - requestValues, 51
  - run, 52
  - runner, 53
  - runSimulations, 52
  - statusBar, 53
  - statusInterval, 53
  - topResults, 53
  - updateStatus, 52
  - valueMap, 53
- controller, 19
  - Optimizer, 86
  - stepState, 19
- Controller::stepstate, 110
  - currentStep, 111
  - get, 110
  - next, 111
  - stepChanged, 111
- coordinate
  - Types.h, 134
- createConfig
  - ConfigEditor, 37
- ctr
  - main.cpp, 131
- currentLevel
  - Levels, 77
- currentStep
  - Controller::stepstate, 111
- D
  - DirectOptimizer, 59
  - HyRect, 74
- definition
  - Parameter, 91
- deleteConfig
  - ConfigEditor, 37
- denormalize
  - ParameterNormalizer, 95
- depth
  - DirectTypes.h, 133
- dimension
  - DirectTypes.h, 133
- DIR
  - ConfigEditor, 40
- dirCoordinate
  - DirectTypes.h, 133
- direct, 19
  - level, 20
- DirectOptimizer, 54
  - activeRects, 59
  - addActiveRects, 56
  - D, 59
  - DirectOptimizer, 56
  - estimatedValue, 56
  - getName, 57
  - getPartitionSize, 57
  - getStatus, 57
  - getStatusBar, 57
  - getValues, 58
  - iterations, 60
  - level, 60
  - normalizer, 60
  - optimalRectangles, 58
  - printValues, 60
  - removeActiveRects, 58
  - runOptimization, 59
  - saveProgress, 59
  - saveValues, 59
  - stopCon, 60
  - trackProgress, 60
- DirectTypes.h
  - depth, 133
  - dimension, 133
  - dirCoordinate, 133
- DiscreteParameter, 60
  - DiscreteParameter, 62
  - getOffset, 63
  - getStep, 63
  - getTimes, 63
  - getVal, 63
  - offset, 64
  - setTimes, 63
  - setVal, 64
  - step, 64
  - times, 64
- divide
  - HyRect, 70
- editor
  - PlexeSimulationRunner, 99
- END\_TIME
  - StoppingCondition, 114
- estimatedValue
  - DirectOptimizer, 56
- evaluate
  - Controller, 51
  - StoppingCondition, 113
  - StubController, 117
- Evaluation, 65
  - getName, 66
  - getStatus, 66
  - getStatusBar, 66
  - processOutput, 66, 67
- evaluation, 22
  - Controller, 52
- exec
  - CommandLine, 35
- f
  - StubController, 118
- functions
  - StubController, 119
- functionValue
  - Types.h, 134



- get
  - Controller::stepstate, 110
- get\_constant\_headway
  - constant\_headway.py, 128
- get\_last\_value
  - constant\_headway.py, 129
- getAvgValue
  - HyRect, 70
- getConfig
  - Parameter, 88
  - ParameterDefinition, 92
- getConfigPath
  - ConfigEditor, 37
- getControllerOption
  - ConfigEditor, 38
- getD
  - HyRect, 70
- getDepth
  - HyRect, 70
- getDiagonalLength
  - HyRect, 71
- getDir
  - ConfigEditor, 38
- getEpsilon
  - Levels, 76
- getIterationsSinceImprov
  - StoppingCondition, 113
- getLevel
  - Levels, 76
- getMax
  - Parameter, 88
  - ParameterDefinition, 93
- getMedian
  - ValueMap, 123
- getMin
  - Parameter, 88
  - ParameterDefinition, 93
- getName
  - ConstantHeadway, 42
  - DirectOptimizer, 57
  - Evaluation, 66
  - Optimizer, 83
  - PlexeSimulationRunner, 98
  - SimulationRunner, 103
  - Status, 105
- getOffset
  - DiscreteParameter, 63
- getPartitionSize
  - DirectOptimizer, 57
- getPos
  - HyRect, 71
- getRectSubset
  - Levels, 76
- getResultPath
  - ConfigEditor, 38
- getRunId
  - PlexeSimulationRunner, 98
- getSamplingVertices
  - BaseRect, 29
  - ChildRect, 31
  - HyRect, 71
- getSize
  - ThreadsafeQueue< Key >, 120
  - ValueMap, 123
- getSplitDim
  - HyRect, 71
- getStartSize
  - ThreadsafeQueue< Key >, 120
- getStatus
  - ConstantHeadway, 42
  - DirectOptimizer, 57
  - Evaluation, 66
  - Optimizer, 84
  - PlexeSimulationRunner, 98
  - SimulationRunner, 103
  - Status, 106
- getStatusBar
  - ConstantHeadway, 43
  - DirectOptimizer, 57
  - Evaluation, 66
  - Optimizer, 84
  - PlexeSimulationRunner, 98
  - SimulationRunner, 103
  - Status, 106
- getStep
  - DiscreteParameter, 63
- getTimes
  - DiscreteParameter, 63
- getTopVals
  - ValueMap, 123
- getUnit
  - Parameter, 88
  - ParameterDefinition, 93
- getVal
  - ContinuousParameter, 46
  - DiscreteParameter, 63
  - Parameter, 89
- getValueMap
  - Controller, 51
  - Optimizer, 84
- getValues
  - DirectOptimizer, 58
  - ValueMap, 123
- global
  - Levels, 77
- GrahamScan, 67
  - scan, 67
- HyRect, 68
  - avgValue, 74
  - D, 74
  - divide, 70
  - getAvgValue, 70
  - getD, 70
  - getDepth, 70
  - getDiagonalLength, 71
  - getPos, 71

- getSamplingVertices, 71
- getSplitDim, 71
- HyRect, 69
- operator!=, 72
- operator<, 72
- operator<=, 72
- operator>, 73
- operator>=, 73
- operator==, 73
- pos, 74
- setAvgValue, 74
- t, 74
- hyrect, 24
  - position, 25
- insert
  - ValueMap, 124
- interruptHandler
  - main.cpp, 130
- isGlobal
  - Levels, 77
- isKnown
  - ValueMap, 124
- isTopValue
  - ValueMap, 124
- iterations
  - DirectOptimizer, 60
- iterationsSinceImprov
  - StoppingCondition, 114
- keepFiles
  - Controller, 53
- L0\_EPSILON
  - Levels, 78
- L0\_SIZE
  - Levels, 78
- L1\_EPSILON
  - Levels, 78
- L1\_SIZE
  - Levels, 78
- L2\_EPSILON
  - Levels, 78
- L2\_SIZE
  - Levels, 78
- L3\_EPSILON
  - Levels, 78
- L3\_SIZE
  - Levels, 79
- LARGE\_DIVIDER
  - StatusBar, 109
- lastStatus
  - StatusBar, 109
- lastStep
  - StatusBar, 109
- lastVal
  - StatusBar, 109
- level
  - direct, 20
  - DirectOptimizer, 60
- Levels, 75
  - currentLevel, 77
  - getEpsilon, 76
  - getLevel, 76
  - getRectSubset, 76
  - global, 77
  - isGlobal, 77
  - L0\_EPSILON, 78
  - L0\_SIZE, 78
  - L1\_EPSILON, 78
  - L1\_SIZE, 78
  - L2\_EPSILON, 78
  - L2\_SIZE, 78
  - L3\_EPSILON, 78
  - L3\_SIZE, 79
  - nextLevel, 77
  - setGlobal, 77
- lowerValues
  - ValueMap, 125
- main
  - main.cpp, 131
- main.cpp
  - ctr, 131
  - interruptHandler, 130
  - main, 131
- max
  - ParameterDefinition, 93
- min
  - ParameterDefinition, 94
- mins
  - StoppingCondition, 115
- Multithreaded
  - Multithreaded< Key, T, Compare, Allocator >, 80
- multithreaded
  - constant\_headway.py, 129
- Multithreaded< Key, T, Compare, Allocator >, 79
  - Multithreaded, 80
  - multithreadFunction, 80
  - NR\_THREADS, 81
  - queue, 81
  - runMultithreadedFunctions, 80
  - work, 81
- multithreadFunction
  - Multithreaded< Key, T, Compare, Allocator >, 80
- next
  - Controller::stepstate, 111
- nextLevel
  - Levels, 77
- NO\_NAME
  - Status, 106
- NO\_STATUS\_SUPPORT
  - Status, 106
- normalize
  - ParameterNormalizer, 95
- normalizer
  - DirectOptimizer, 60

- NR\_ACCURACY\_ITERATIONS
  - StoppingCondition, 115
- NR\_EVALUATIONS
  - StoppingCondition, 115
- NR\_HYRECTS
  - StoppingCondition, 115
- NR\_THREADS
  - ConstantHeadway, 44
  - Multithreaded< Key, T, Compare, Allocator >, 81
- offset
  - DiscreteParameter, 64
- operationsLock
  - ValueMap, 125
- operator!=
  - HyRect, 72
  - Parameter, 89
- operator<
  - HyRect, 72
  - Parameter, 89
- operator<=
  - HyRect, 72
  - Parameter, 90
- operator>
  - HyRect, 73
  - Parameter, 90
- operator>=
  - HyRect, 73
  - Parameter, 91
- operator()
  - CmpPairVectorSharedParameterFunctionvalue, 32
  - CmpPtrFunctionvalue, 33
  - CmpSharedHyrect, 34
  - CmpVectorSharedParameter, 34
- operator==
  - ChildRect, 31
  - HyRect, 73
  - Parameter, 90
- optimalRectangles
  - DirectOptimizer, 58
- Optimizer, 82
  - controller, 86
  - getName, 83
  - getStatus, 84
  - getStatusBar, 84
  - getValueMap, 84
  - Optimizer, 83
  - parameters, 86
  - requestValues, 84
  - runOptimization, 86
- optimizer, 24
  - Controller, 53
- Parameter, 86
  - definition, 91
  - getConfig, 88
  - getMax, 88
  - getMin, 88
  - getUnit, 88
  - getVal, 89
  - operator!=, 89
  - operator<, 89
  - operator<=, 90
  - operator>, 90
  - operator>=, 91
  - operator==, 90
  - Parameter, 88
  - setVal, 91
- ParameterDefinition, 91
  - config, 93
  - getConfig, 92
  - getMax, 93
  - getMin, 93
  - getUnit, 93
  - max, 93
  - min, 94
  - ParameterDefinition, 92
  - unit, 94
- ParameterNormalizer, 94
  - denormalize, 95
  - normalize, 95
  - ParameterNormalizer, 95
  - parameters, 95
- parameters, 22
  - Optimizer, 86
  - ParameterNormalizer, 95
- parent
  - ChildRect, 32
- pFunc
  - PythonScript, 101
- plex, 21
- PlexeSimulationRunner, 96
  - editor, 99
  - getName, 98
  - getRunId, 98
  - getStatus, 98
  - getStatusBar, 98
  - PlexeSimulationRunner, 97
  - REPEAT, 99
  - runNumber, 99
  - runNumberLock, 100
  - SCENARIOS, 100
  - work, 98, 99
- pModule
  - PythonScript, 101
- pop
  - ThreadsafeQueue< Key >, 120
- pos
  - HyRect, 74
- position
  - hyrect, 25
- printResult
  - StatusBar, 108
- printResults
  - StatusBar, 108
- printStatus
  - StatusBar, 108

- printValues
  - DirectOptimizer, 60
- processOutput
  - ConstantHeadway, 43
  - Evaluation, 66, 67
- push
  - ThreadsafeQueue< Key >, 120
- PythonScript, 100
  - ~PythonScript, 101
  - pFunc, 101
  - pModule, 101
  - PythonScript, 101
- query
  - ValueMap, 125
- queue
  - Multithreaded< Key, T, Compare, Allocator >, 81
- queueLock
  - ThreadsafeQueue< Key >, 120
- removeActiveRects
  - DirectOptimizer, 58
- removeOldResultfiles
  - Controller, 51
  - StubController, 118
- REPEAT
  - PlexeSimulationRunner, 99
- replaceOption
  - ConfigEditor, 39
- requestValues
  - Controller, 51
  - Optimizer, 84
- RESULTS
  - ConfigEditor, 40
- run
  - Controller, 52
- runId
  - Types.h, 134
- runMultithreadedFunctions
  - Multithreaded< Key, T, Compare, Allocator >, 80
- runner, 23
  - Controller, 53
- runNumber
  - PlexeSimulationRunner, 99
- runNumberLock
  - PlexeSimulationRunner, 100
- runOptimization
  - DirectOptimizer, 59
  - Optimizer, 86
- runSimulations
  - Controller, 52
  - SimulationRunner, 104
  - StubController, 118
- safeQueue
  - ThreadsafeQueue< Key >, 121
- saveProgress
  - DirectOptimizer, 59
- saveValues
  - DirectOptimizer, 59
- scan
  - GrahamScan, 67
- SCENARIOS
  - PlexeSimulationRunner, 100
- secureValue
  - ConstantHeadway, 44
- setAvgValue
  - HyRect, 74
- setGlobal
  - Levels, 77
- setResultFiles
  - ConfigEditor, 39
- setStartNow
  - StoppingCondition, 113
- setTimes
  - DiscreteParameter, 63
- setVal
  - ContinuousParameter, 46
  - DiscreteParameter, 64
  - Parameter, 91
- SimulationRunner, 102
  - getName, 103
  - getStatus, 103
  - getStatusBar, 103
  - runSimulations, 104
  - SimulationRunner, 103
  - work, 104
- SMALL\_DIVIDER
  - StatusBar, 110
- startSize
  - ThreadsafeQueue< Key >, 121
- Status, 105
  - getName, 105
  - getStatus, 106
  - getStatusBar, 106
  - NO\_NAME, 106
  - NO\_STATUS\_SUPPORT, 106
- status, 23
  - step, 24
- StatusBar, 107
  - LARGE\_DIVIDER, 109
  - lastStatus, 109
  - lastStep, 109
  - lastVal, 109
  - printResult, 108
  - printResults, 108
  - printStatus, 108
  - SMALL\_DIVIDER, 110
  - updateStatus, 108
- statusBar
  - Controller, 53
- statusInterval
  - Controller, 53
- step
  - DiscreteParameter, 64
  - status, 24
- stepChanged

- Controller::stepstate, 111
- stepState
  - controller, 19
- stopCon
  - DirectOptimizer, 60
- StoppingCondition, 111
  - ACCURACY, 114
  - bestVal, 114
  - END\_TIME, 114
  - evaluate, 113
  - getIterationsSinceImprov, 113
  - iterationsSinceImprov, 114
  - mins, 115
  - NR\_ACCURACY\_ITERATIONS, 115
  - NR\_EVALUATIONS, 115
  - NR\_HYRECTS, 115
  - setStartNow, 113
  - StoppingCondition, 112, 113
  - time\_eval, 115
  - updateAccuracy, 114
- StubController, 115
  - evaluate, 117
  - f, 118
  - functions, 119
  - removeOldResultfiles, 118
  - runSimulations, 118
  - StubController, 117
  - updateStatus, 118
- t
  - HyRect, 74
- tba
  - ValueMap, 125
- ThreadsafeQueue< Key >, 119
  - getSize, 120
  - getStartSize, 120
  - pop, 120
  - push, 120
  - queueLock, 120
  - safeQueue, 121
  - startSize, 121
- time\_eval
  - StoppingCondition, 115
- times
  - DiscreteParameter, 64
- topEntries
  - ValueMap, 126
- topResults
  - Controller, 53
- topVals
  - ValueMap, 126
- trackProgress
  - DirectOptimizer, 60
- Types.h
  - coordinate, 134
  - functionValue, 134
  - runId, 134
- unit
  - ParameterDefinition, 94
  - updateAccuracy
    - StoppingCondition, 114
  - updateMap
    - ValueMap, 125
  - updateStatus
    - Controller, 52
    - StatusBar, 108
    - StubController, 118
  - upperValues
    - ValueMap, 126
  - usedThreads
    - ConstantHeadway, 44
  - utils, 25
  - val
    - ContinuousParameter, 48
  - ValueMap, 121
    - addValue, 122
    - getMedian, 123
    - getSize, 123
    - getTopVals, 123
    - getValues, 123
    - insert, 124
    - isKnown, 124
    - isTopValue, 124
    - lowerValues, 125
    - operationsLock, 125
    - query, 125
    - tba, 125
    - topEntries, 126
    - topVals, 126
    - updateMap, 125
    - upperValues, 126
    - ValueMap, 122
    - values, 126
  - valueMap
    - Controller, 53
  - values
    - ValueMap, 126
  - work
    - Multithreaded< Key, T, Compare, Allocator >, 81
    - PlexeSimulationRunner, 98, 99
    - SimulationRunner, 104