

Simopticon

1.1.0

Generated by Doxygen 1.9.1

1 Documentation	1
1.1 Overview	1
1.2 Setup	2
1.2.1 Requirements	2
1.2.2 Installation	2
1.2.3 Update	3
1.3 Usage	3
1.3.1 Configuration	3
1.3.2 Available Optimizers	3
1.3.3 Optimization	4
1.4 Extension	4
1.4.1 Development	5
1.4.2 Integration	5
2 Todo List	7
3 Module Index	9
3.1 Modules	9
4 Hierarchical Index	11
4.1 Class Hierarchy	11
5 Class Index	13
5.1 Class List	13
6 File Index	15
6.1 File List	15
7 Module Documentation	19
7.1 controller	19
7.1.1 Detailed Description	19
7.1.2 Variable Documentation	19
7.1.2.1 stepState	19
7.2 direct	19
7.2.1 Detailed Description	20
7.2.2 Enumeration Type Documentation	21
7.2.2.1 level	21
7.3 montecarlo	21
7.3.1 Detailed Description	21
7.4 randomneighbors	21
7.4.1 Detailed Description	22
7.5 plexe	22
7.5.1 Detailed Description	22
7.6 constant_headway	22

7.6.1 Detailed Description	23
7.7 evaluation	23
7.7.1 Detailed Description	23
7.8 optimizer	23
7.8.1 Detailed Description	24
7.9 hyrect	24
7.9.1 Detailed Description	25
7.9.2 Enumeration Type Documentation	25
7.9.2.1 position	25
7.10 parameters	25
7.10.1 Detailed Description	25
7.11 status	25
7.11.1 Detailed Description	26
7.11.2 Enumeration Type Documentation	26
7.11.2.1 step	26
7.12 runner	26
7.12.1 Detailed Description	27
7.13 utils	27
7.13.1 Detailed Description	27
8 Class Documentation	29
8.1 Abortable Class Reference	29
8.1.1 Detailed Description	30
8.1.2 Member Function Documentation	30
8.1.2.1 abort()	30
8.1.3 Member Data Documentation	30
8.1.3.1 aborted	30
8.2 BaseRect Class Reference	30
8.2.1 Detailed Description	31
8.2.2 Constructor & Destructor Documentation	31
8.2.2.1 BaseRect()	31
8.2.3 Member Function Documentation	31
8.2.3.1 getSamplingVertices()	32
8.3 ChildRect Class Reference	32
8.3.1 Detailed Description	33
8.3.2 Constructor & Destructor Documentation	33
8.3.2.1 ChildRect()	33
8.3.3 Member Function Documentation	33
8.3.3.1 getSamplingVertices()	33
8.3.3.2 operator==()	34
8.3.4 Member Data Documentation	34
8.3.4.1 parent	34

8.4 CmpPairVectorSharedParameterFunctionvalue Struct Reference	34
8.4.1 Detailed Description	35
8.4.2 Member Function Documentation	35
8.4.2.1 operator()	35
8.5 CmpPtrFunctionvalue Struct Reference	35
8.5.1 Detailed Description	36
8.5.2 Member Function Documentation	36
8.5.2.1 operator()	36
8.6 CmpSharedHyrect Struct Reference	36
8.6.1 Detailed Description	36
8.6.2 Member Function Documentation	36
8.6.2.1 operator()	37
8.7 CmpVectorSharedParameter Struct Reference	37
8.7.1 Detailed Description	37
8.7.2 Member Function Documentation	37
8.7.2.1 operator()	37
8.8 CommandLine Class Reference	38
8.8.1 Detailed Description	38
8.8.2 Member Function Documentation	38
8.8.2.1 exec()	38
8.9 ConfigEditor Class Reference	39
8.9.1 Detailed Description	40
8.9.2 Constructor & Destructor Documentation	40
8.9.2.1 ConfigEditor()	40
8.9.3 Member Function Documentation	40
8.9.3.1 createConfig()	40
8.9.3.2 deleteConfig()	41
8.9.3.3 getConfigPath()	41
8.9.3.4 getControllerOption()	41
8.9.3.5 getDir()	42
8.9.3.6 getResultPath()	42
8.9.3.7 replaceOption() [1/2]	42
8.9.3.8 replaceOption() [2/2]	42
8.9.3.9 setResultFiles()	43
8.9.4 Member Data Documentation	43
8.9.4.1 CONFIG	43
8.9.4.2 CONTROLLER	43
8.9.4.3 DIR	43
8.9.4.4 RESULTS	43
8.10 ConstantHeadway Class Reference	44
8.10.1 Detailed Description	45
8.10.2 Constructor & Destructor Documentation	45

8.10.2.1 ConstantHeadway()	45
8.10.3 Member Function Documentation	45
8.10.3.1 getName()	46
8.10.3.2 getStatus()	46
8.10.3.3 getStatusBar()	46
8.10.3.4 processOutput() [1/2]	46
8.10.3.5 processOutput() [2/2]	47
8.10.3.6 secureValue()	47
8.10.4 Member Data Documentation	47
8.10.4.1 NR_THREADS	47
8.10.4.2 usedThreads	47
8.11 ContinuousParameter Class Reference	48
8.11.1 Detailed Description	49
8.11.2 Constructor & Destructor Documentation	49
8.11.2.1 ContinuousParameter() [1/2]	49
8.11.2.2 ContinuousParameter() [2/2]	49
8.11.3 Member Function Documentation	49
8.11.3.1 getVal()	49
8.11.3.2 setVal()	50
8.11.4 Member Data Documentation	50
8.11.4.1 val	50
8.12 Controller Class Reference	50
8.12.1 Detailed Description	52
8.12.2 Constructor & Destructor Documentation	53
8.12.2.1 Controller()	53
8.12.3 Member Function Documentation	53
8.12.3.1 abort()	53
8.12.3.2 evaluate()	53
8.12.3.3 getValueMap()	54
8.12.3.4 removeOldResultfiles()	54
8.12.3.5 requestValues()	54
8.12.3.6 run()	54
8.12.3.7 runSimulations()	54
8.12.3.8 saveValues()	55
8.12.3.9 updateStatus()	55
8.12.4 Member Data Documentation	55
8.12.4.1 evaluation	55
8.12.4.2 keepFiles	55
8.12.4.3 optimizer	56
8.12.4.4 printValues	56
8.12.4.5 runner	56
8.12.4.6 statusBar	56

8.12.4.7 statusInterval	56
8.12.4.8 topResults	56
8.12.4.9 valueMap	56
8.13 DirectOptimizer Class Reference	57
8.13.1 Detailed Description	58
8.13.2 Constructor & Destructor Documentation	58
8.13.2.1 DirectOptimizer()	59
8.13.3 Member Function Documentation	59
8.13.3.1 addActiveRects()	59
8.13.3.2 estimatedValue()	59
8.13.3.3 getName()	60
8.13.3.4 getPartitionSize()	60
8.13.3.5 getStatus()	60
8.13.3.6 getStatusBar()	60
8.13.3.7 getValues()	60
8.13.3.8 optimalRectangles()	61
8.13.3.9 removeActiveRects()	61
8.13.3.10 runOptimization()	61
8.13.3.11 saveProgress()	62
8.13.4 Member Data Documentation	62
8.13.4.1 activeRects	62
8.13.4.2 D	62
8.13.4.3 iterations	62
8.13.4.4 level	62
8.13.4.5 normalizer	62
8.13.4.6 stopCon	63
8.13.4.7 trackProgress	63
8.14 DirectStoppingCondition Class Reference	63
8.14.1 Detailed Description	64
8.14.2 Constructor & Destructor Documentation	64
8.14.2.1 DirectStoppingCondition() [1/2]	64
8.14.2.2 DirectStoppingCondition() [2/2]	64
8.14.3 Member Function Documentation	65
8.14.3.1 evaluate()	65
8.14.4 Member Data Documentation	65
8.14.4.1 NR_HYRECTS	65
8.15 DiscreteParameter Class Reference	65
8.15.1 Detailed Description	67
8.15.2 Constructor & Destructor Documentation	67
8.15.2.1 DiscreteParameter() [1/2]	67
8.15.2.2 DiscreteParameter() [2/2]	67
8.15.3 Member Function Documentation	68

8.15.3.1	getOffset()	68
8.15.3.2	getStep()	68
8.15.3.3	getTimes()	68
8.15.3.4	getVal()	68
8.15.3.5	setTimes()	68
8.15.3.6	setVal()	69
8.15.4	Member Data Documentation	69
8.15.4.1	offset	69
8.15.4.2	step	69
8.15.4.3	times	69
8.16	Evaluation Class Reference	70
8.16.1	Detailed Description	71
8.16.2	Member Function Documentation	71
8.16.2.1	getName()	71
8.16.2.2	getStatus()	71
8.16.2.3	getStatusBar()	71
8.16.2.4	processOutput() [1/2]	72
8.16.2.5	processOutput() [2/2]	72
8.17	GrahamScan Class Reference	72
8.17.1	Detailed Description	73
8.17.2	Member Function Documentation	73
8.17.2.1	scan()	73
8.18	HyRect Class Reference	73
8.18.1	Detailed Description	75
8.18.2	Constructor & Destructor Documentation	75
8.18.2.1	HyRect()	75
8.18.3	Member Function Documentation	76
8.18.3.1	divide()	76
8.18.3.2	getAvgValue()	76
8.18.3.3	getD()	76
8.18.3.4	getDepth()	76
8.18.3.5	getDiagonalLength()	77
8.18.3.6	getPos()	77
8.18.3.7	getSamplingVertices()	77
8.18.3.8	getSplitDim()	77
8.18.3.9	operator!=(())	77
8.18.3.10	operator<()	78
8.18.3.11	operator<=()	78
8.18.3.12	operator==(())	78
8.18.3.13	operator>()	79
8.18.3.14	operator>=()	79
8.18.3.15	setAvgValue()	79

8.18.4 Member Data Documentation	80
8.18.4.1 avgValue	80
8.18.4.2 D	80
8.18.4.3 pos	80
8.18.4.4 t	80
8.19 Levels Class Reference	80
8.19.1 Detailed Description	82
8.19.2 Member Function Documentation	82
8.19.2.1 getEpsilon()	82
8.19.2.2 getLevel()	82
8.19.2.3 getRectSubset()	82
8.19.2.4 isGlobal()	83
8.19.2.5 nextLevel()	83
8.19.2.6 setGlobal()	83
8.19.3 Member Data Documentation	83
8.19.3.1 currentLevel	83
8.19.3.2 global	84
8.19.3.3 L0_EPSILON	84
8.19.3.4 L0_SIZE	84
8.19.3.5 L1_EPSILON	84
8.19.3.6 L1_SIZE	84
8.19.3.7 L2_EPSILON	84
8.19.3.8 L2_SIZE	84
8.19.3.9 L3_EPSILON	84
8.19.3.10 L3_SIZE	85
8.20 MonteCarlo Class Reference	85
8.20.1 Detailed Description	87
8.20.2 Constructor & Destructor Documentation	87
8.20.2.1 MonteCarlo()	87
8.20.3 Member Function Documentation	87
8.20.3.1 getName()	87
8.20.3.2 getStatus()	87
8.20.3.3 getStatusBar()	88
8.20.3.4 runOptimization()	88
8.20.3.5 saveProgress()	88
8.20.4 Member Data Documentation	88
8.20.4.1 iterations	88
8.20.4.2 lastEvaluations	89
8.20.4.3 parallelTrials	89
8.20.4.4 stopCon	89
8.20.4.5 trackProgress	89
8.21 Multithreaded< Key, T, Compare, Allocator > Class Template Reference	89

8.21.1 Detailed Description	91
8.21.2 Constructor & Destructor Documentation	91
8.21.2.1 Multithreaded()	91
8.21.3 Member Function Documentation	91
8.21.3.1 multithreadFunction()	92
8.21.3.2 runMultithreadedFunctions()	92
8.21.3.3 work()	92
8.21.4 Member Data Documentation	93
8.21.4.1 NR_THREADS	93
8.21.4.2 queue	93
8.22 Optimizer Class Reference	93
8.22.1 Detailed Description	95
8.22.2 Constructor & Destructor Documentation	95
8.22.2.1 Optimizer()	95
8.22.3 Member Function Documentation	95
8.22.3.1 getName()	95
8.22.3.2 getStatus()	95
8.22.3.3 getStatusBar()	96
8.22.3.4 getValueMap()	96
8.22.3.5 requestValues()	96
8.22.3.6 runOptimization()	96
8.22.4 Member Data Documentation	97
8.22.4.1 controller	97
8.22.4.2 parameters	97
8.23 Parameter Class Reference	97
8.23.1 Detailed Description	98
8.23.2 Constructor & Destructor Documentation	99
8.23.2.1 Parameter()	99
8.23.3 Member Function Documentation	99
8.23.3.1 getConfig()	99
8.23.3.2 getMax()	99
8.23.3.3 getMin()	99
8.23.3.4 getUnit()	100
8.23.3.5 getVal()	100
8.23.3.6 operator!=(())	100
8.23.3.7 operator<()	100
8.23.3.8 operator<=()	101
8.23.3.9 operator==(())	101
8.23.3.10 operator>()	101
8.23.3.11 operator>=()	102
8.23.3.12 setVal()	102
8.23.4 Member Data Documentation	102

8.23.4.1 definition	102
8.24 ParameterDefinition Class Reference	102
8.24.1 Detailed Description	103
8.24.2 Constructor & Destructor Documentation	103
8.24.2.1 ParameterDefinition()	103
8.24.3 Member Function Documentation	104
8.24.3.1 getConfig()	104
8.24.3.2 getMax()	104
8.24.3.3 getMin()	104
8.24.3.4 getUnit()	104
8.24.4 Member Data Documentation	105
8.24.4.1 config	105
8.24.4.2 max	105
8.24.4.3 min	105
8.24.4.4 unit	105
8.25 ParameterNormalizer Class Reference	105
8.25.1 Detailed Description	106
8.25.2 Constructor & Destructor Documentation	106
8.25.2.1 ParameterNormalizer()	106
8.25.3 Member Function Documentation	106
8.25.3.1 denormalize()	107
8.25.3.2 normalize()	107
8.25.4 Member Data Documentation	107
8.25.4.1 parameters	107
8.26 PlexeSimulationRunner Class Reference	107
8.26.1 Detailed Description	109
8.26.2 Constructor & Destructor Documentation	109
8.26.2.1 PlexeSimulationRunner()	110
8.26.3 Member Function Documentation	110
8.26.3.1 getName()	110
8.26.3.2 getRunId()	110
8.26.3.3 getStatus()	110
8.26.3.4 getStatusBar()	111
8.26.3.5 work() [1/2]	111
8.26.3.6 work() [2/2]	111
8.26.4 Member Data Documentation	112
8.26.4.1 editor	112
8.26.4.2 REPEAT	112
8.26.4.3 runNumber	112
8.26.4.4 runNumberLock	112
8.26.4.5 SCENARIOS	112
8.27 PythonScript Class Reference	112

8.27.1 Detailed Description	113
8.27.2 Constructor & Destructor Documentation	113
8.27.2.1 PythonScript()	114
8.27.2.2 ~PythonScript()	114
8.27.3 Member Data Documentation	114
8.27.3.1 pFunc	114
8.27.3.2 pModule	114
8.28 RandomNeighbors Class Reference	114
8.28.1 Detailed Description	116
8.28.2 Constructor & Destructor Documentation	116
8.28.2.1 RandomNeighbors()	116
8.28.3 Member Function Documentation	117
8.28.3.1 getName()	117
8.28.3.2 getStatus()	117
8.28.3.3 getStatusBar()	117
8.28.3.4 runOptimization()	118
8.28.3.5 saveProgress()	118
8.28.4 Member Data Documentation	118
8.28.4.1 iterations	118
8.28.4.2 lastEvaluations	118
8.28.4.3 lastLocal	118
8.28.4.4 localSearchProbability	118
8.28.4.5 neighborhoodWidth	119
8.28.4.6 parallelTrials	119
8.28.4.7 stopCon	119
8.28.4.8 trackProgress	119
8.29 SimulationRunner Class Reference	119
8.29.1 Detailed Description	121
8.29.2 Constructor & Destructor Documentation	122
8.29.2.1 SimulationRunner()	122
8.29.3 Member Function Documentation	122
8.29.3.1 getName()	122
8.29.3.2 getStatus()	122
8.29.3.3 getStatusBar()	122
8.29.3.4 runSimulations()	123
8.29.3.5 work()	123
8.30 Status Class Reference	123
8.30.1 Detailed Description	124
8.30.2 Member Function Documentation	124
8.30.2.1 getName()	125
8.30.2.2 getStatus()	125
8.30.2.3 getStatusBar()	125

8.30.3 Member Data Documentation	125
8.30.3.1 NO_NAME	125
8.30.3.2 NO_STATUS_SUPPORT	126
8.31 StatusBar Class Reference	126
8.31.1 Detailed Description	127
8.31.2 Member Function Documentation	127
8.31.2.1 printResult()	127
8.31.2.2 printResults()	127
8.31.2.3 printStatus()	127
8.31.2.4 updateStatus()	128
8.31.3 Member Data Documentation	128
8.31.3.1 LARGE_DIVIDER	128
8.31.3.2 lastStatus	128
8.31.3.3 lastStep	129
8.31.3.4 lastVal	129
8.31.3.5 SMALL_DIVIDER	129
8.32 Controller::stepstate Struct Reference	129
8.32.1 Detailed Description	130
8.32.2 Member Function Documentation	130
8.32.2.1 get()	130
8.32.2.2 next()	130
8.32.3 Member Data Documentation	130
8.32.3.1 currentStep	130
8.32.3.2 stepChanged	130
8.33 StoppingCondition Class Reference	131
8.33.1 Detailed Description	132
8.33.2 Constructor & Destructor Documentation	132
8.33.2.1 StoppingCondition() [1/2]	132
8.33.2.2 StoppingCondition() [2/2]	133
8.33.3 Member Function Documentation	133
8.33.3.1 evaluate()	133
8.33.3.2 getConditionFromJSON()	133
8.33.3.3 getIterationsSinceImprov()	134
8.33.3.4 setStartNow()	134
8.33.3.5 updateAccuracy()	134
8.33.4 Member Data Documentation	134
8.33.4.1 ACCURACY	135
8.33.4.2 bestVal	135
8.33.4.3 END_TIME	135
8.33.4.4 iterationsSinceImprov	135
8.33.4.5 mins	135
8.33.4.6 NR_ACCURACY_ITERATIONS	135

8.33.4.7 NR_EVALUATIONS	135
8.33.4.8 time_eval	136
8.34 StubController Class Reference	136
8.34.1 Detailed Description	137
8.34.2 Constructor & Destructor Documentation	138
8.34.2.1 StubController()	138
8.34.3 Member Function Documentation	138
8.34.3.1 evaluate()	138
8.34.3.2 removeOldResultfiles()	138
8.34.3.3 runSimulations()	139
8.34.3.4 updateStatus()	139
8.34.4 Member Data Documentation	139
8.34.4.1 f	139
8.34.4.2 functions	139
8.35 ThreadsafeQueue< Key > Class Template Reference	140
8.35.1 Detailed Description	140
8.35.2 Member Function Documentation	140
8.35.2.1 getSize()	141
8.35.2.2 getStartSize()	141
8.35.2.3 pop()	141
8.35.2.4 push()	141
8.35.3 Member Data Documentation	141
8.35.3.1 queueLock	142
8.35.3.2 safeQueue	142
8.35.3.3 startSize	142
8.36 ValueMap Class Reference	142
8.36.1 Detailed Description	143
8.36.2 Constructor & Destructor Documentation	143
8.36.2.1 ValueMap()	144
8.36.3 Member Function Documentation	144
8.36.3.1 addValue()	144
8.36.3.2 getMedian()	144
8.36.3.3 getSize()	144
8.36.3.4 getTopVals()	145
8.36.3.5 getValues()	145
8.36.3.6 insert()	145
8.36.3.7 isKnown()	145
8.36.3.8 isTopValue()	146
8.36.3.9 query()	146
8.36.3.10 updateMap()	146
8.36.4 Member Data Documentation	146
8.36.4.1 lowerValues	147

8.36.4.2 operationsLock	147
8.36.4.3 tba	147
8.36.4.4 topEntries	147
8.36.4.5 topVals	147
8.36.4.6 upperValues	147
8.36.4.7 values	147
9 File Documentation	149
9.1 /home/runner/work/Simopticon/Simopticon/src/ComparisonFunctions.h File Reference	149
9.1.1 Detailed Description	150
9.2 /home/runner/work/Simopticon/Simopticon/src/controller/Controller.cpp File Reference	150
9.2.1 Detailed Description	151
9.2.2 Function Documentation	151
9.2.2.1 getConfigByPath()	151
9.3 /home/runner/work/Simopticon/Simopticon/src/controller/Controller.h File Reference	151
9.3.1 Detailed Description	152
9.4 /home/runner/work/Simopticon/Simopticon/src/controller/StubController.cpp File Reference	152
9.4.1 Detailed Description	153
9.4.2 Function Documentation	153
9.4.2.1 hartman()	153
9.4.2.2 shekel()	153
9.5 /home/runner/work/Simopticon/Simopticon/src/controller/StubController.h File Reference	154
9.5.1 Detailed Description	154
9.6 /home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.cpp File Reference	155
9.6.1 Detailed Description	155
9.7 /home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.h File Reference	155
9.7.1 Detailed Description	156
9.8 /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/constant_headway.py File Reference	156
9.8.1 Detailed Description	157
9.8.2 Function Documentation	157
9.8.2.1 get_constant_headway()	157
9.8.2.2 multithreaded()	157
9.9 /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.cpp File Reference	158
9.9.1 Detailed Description	158
9.10 /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.h File Reference	158
9.10.1 Detailed Description	159
9.11 /home/runner/work/Simopticon/Simopticon/src/evaluation/Evaluation.cpp File Reference	159
9.11.1 Detailed Description	160
9.12 /home/runner/work/Simopticon/Simopticon/src/evaluation/Evaluation.h File Reference	160
9.12.1 Detailed Description	161

9.13	/home/runner/work/Simopticon/Simopticon/src/main.cpp File Reference	161
9.13.1	Detailed Description	162
9.13.2	Function Documentation	162
9.13.2.1	interruptHandler()	162
9.13.2.2	main()	162
9.13.3	Variable Documentation	162
9.13.3.1	ctr	163
9.14	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectComparisonFunctions.h File Reference	163
9.14.1	Detailed Description	164
9.15	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectOptimizer.cpp File Reference	164
9.15.1	Detailed Description	164
9.16	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectOptimizer.h File Reference	164
9.16.1	Detailed Description	165
9.17	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectStoppingCondition.cpp File Reference	165
9.17.1	Detailed Description	166
9.18	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectStoppingCondition.h File Reference	166
9.18.1	Detailed Description	167
9.19	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectTypes.h File Reference	167
9.19.1	Detailed Description	168
9.19.2	Typedef Documentation	168
9.19.2.1	depth	168
9.19.2.2	dimension	168
9.19.2.3	dirCoordinate	168
9.20	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/GrahamScan.cpp File Reference	169
9.20.1	Detailed Description	169
9.21	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/GrahamScan.h File Reference	169
9.21.1	Detailed Description	170
9.22	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/BaseRect.cpp File Reference	170
9.22.1	Detailed Description	170
9.23	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/BaseRect.h File Reference	171
9.23.1	Detailed Description	171
9.24	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ChildRect.cpp File Reference	172
9.24.1	Detailed Description	172
9.25	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ChildRect.h File Reference	172
9.25.1	Detailed Description	173
9.26	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/HyRect.cpp File Reference	173
9.26.1	Detailed Description	173
9.27	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/HyRect.h File Reference	174
9.27.1	Detailed Description	175
9.28	/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Levels.cpp File Reference	175

9.28.1 Detailed Description	175
9.29 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Levels.h File Reference	175
9.29.1 Detailed Description	176
9.30 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ParameterNormalizer.cpp File Reference	177
9.30.1 Detailed Description	177
9.31 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ParameterNormalizer.h File Reference	177
9.31.1 Detailed Description	178
9.32 /home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/MonteCarlo.cpp File Reference	179
9.32.1 Detailed Description	179
9.33 /home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/MonteCarlo.h File Reference	179
9.33.1 Detailed Description	180
9.34 /home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.cpp File Reference	180
9.34.1 Detailed Description	180
9.35 /home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.h File Reference	180
9.35.1 Detailed Description	181
9.36 /home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors/RandomNeighbors.cpp File Reference	181
9.36.1 Detailed Description	182
9.37 /home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors/RandomNeighbors.h File Reference	182
9.37.1 Detailed Description	183
9.38 /home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.cpp File Reference	183
9.38.1 Detailed Description	183
9.39 /home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.h File Reference	184
9.39.1 Detailed Description	184
9.40 /home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousParameter.cpp File Reference	185
9.40.1 Detailed Description	185
9.41 /home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousParameter.h File Reference	186
9.41.1 Detailed Description	186
9.42 /home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParameter.cpp File Reference	187
9.42.1 Detailed Description	187
9.43 /home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParameter.h File Reference	188
9.43.1 Detailed Description	189
9.44 /home/runner/work/Simopticon/Simopticon/src/parameters/Parameter.cpp File Reference	189
9.44.1 Detailed Description	189
9.45 /home/runner/work/Simopticon/Simopticon/src/parameters/Parameter.h File Reference	189
9.45.1 Detailed Description	190
9.46 /home/runner/work/Simopticon/Simopticon/src/parameters/ParameterDefinition.cpp File Reference	190
9.46.1 Detailed Description	191
9.47 /home/runner/work/Simopticon/Simopticon/src/parameters/ParameterDefinition.h File Reference	191
9.47.1 Detailed Description	192
9.48 /home/runner/work/Simopticon/Simopticon/src/runner/plex/ConfigEditor.cpp File Reference	192

9.48.1 Detailed Description	193
9.49 /home/runner/work/Simopticon/Simopticon/src/runner/plex/ConfigEditor.h File Reference	193
9.49.1 Detailed Description	194
9.50 /home/runner/work/Simopticon/Simopticon/src/runner/plex/PlexeSimulationRunner.cpp File Reference	194
9.50.1 Detailed Description	195
9.51 /home/runner/work/Simopticon/Simopticon/src/runner/plex/PlexeSimulationRunner.h File Reference	195
9.51.1 Detailed Description	196
9.52 /home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.cpp File Reference	196
9.52.1 Detailed Description	196
9.53 /home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.h File Reference	196
9.53.1 Detailed Description	197
9.54 /home/runner/work/Simopticon/Simopticon/src/status/Status.cpp File Reference	197
9.54.1 Detailed Description	198
9.55 /home/runner/work/Simopticon/Simopticon/src/status/Status.h File Reference	198
9.55.1 Detailed Description	199
9.56 /home/runner/work/Simopticon/Simopticon/src/status/StatusBar.cpp File Reference	199
9.56.1 Detailed Description	199
9.57 /home/runner/work/Simopticon/Simopticon/src/status/StatusBar.h File Reference	199
9.57.1 Detailed Description	200
9.58 /home/runner/work/Simopticon/Simopticon/src/Types.h File Reference	201
9.58.1 Detailed Description	201
9.58.2 Typedef Documentation	201
9.58.2.1 coordinate	202
9.58.2.2 functionValue	202
9.58.2.3 parameterCombination	202
9.58.2.4 runId	202
9.59 /home/runner/work/Simopticon/Simopticon/src/utis/Abortable.cpp File Reference	202
9.59.1 Detailed Description	202
9.60 /home/runner/work/Simopticon/Simopticon/src/utis/Abortable.h File Reference	203
9.60.1 Detailed Description	203
9.61 /home/runner/work/Simopticon/Simopticon/src/utis/CommandLine.cpp File Reference	203
9.61.1 Detailed Description	204
9.62 /home/runner/work/Simopticon/Simopticon/src/utis/CommandLine.h File Reference	204
9.62.1 Detailed Description	204
9.63 /home/runner/work/Simopticon/Simopticon/src/utis/Multithreaded.h File Reference	205
9.63.1 Detailed Description	205
9.64 /home/runner/work/Simopticon/Simopticon/src/utis/Multithreaded.tpp File Reference	205
9.64.1 Detailed Description	206
9.65 /home/runner/work/Simopticon/Simopticon/src/utis/PythonScript.cpp File Reference	206
9.65.1 Detailed Description	207
9.66 /home/runner/work/Simopticon/Simopticon/src/utis/PythonScript.h File Reference	207

9.66.1 Detailed Description	208
9.67 /home/runner/work/Simopticon/Simopticon/src/Utils/ThreadSafeQueue.h File Reference	208
9.67.1 Detailed Description	209
9.68 /home/runner/work/Simopticon/Simopticon/src/Utils/ThreadSafeQueue.hpp File Reference	209
9.68.1 Detailed Description	210
Index	211

Chapter 1

Documentation

-
1. [Overview](#)
 2. [Setup](#)
 - (a) [Requirements](#)
 - (b) [Installation](#)
 - (c) [Update](#)
 3. [Usage](#)
 - (a) [Configuration](#)
 - (b) [Available Optimizers](#)
 - (c) [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 **SimulationRunner** and **Evaluation**, tailored for the optimization of platoon controllers using the **Plexe** framework. The available Optimizers are explained in [Available Optimizers](#).

The full API documentation may be found on peternaggschga.github.io/Simopticon or in the comprehensive [PDF file](#) provided. A more in-depth explanation of *Simopticon* and its design principles may be found in the [german bachelor's thesis](#) that proposed the framework.

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](#)).

1.2.3 Update

To upgrade to the latest version of *Simopticon*, the latest release must be pulled and recompiled. Go to the directory, you installed *Simopticon* in, e.g. `~/src/simopticon`. Then pull from `master` using:

```
git pull
```

Go to the `build` directory and rebuild the executable by calling:

```
cmake ..
make -j $(nproc)
```

The resulting `simopticon` executable file contains the latest version of *Simopticon*.

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/plex.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]/plex/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 Available Optimizers

Simopticon contains implementations of multiple optimization strategies, which are shortly described here. Which algorithm is used can be selected in the main config (`config/simopticon.json`).

DIRECT-Algorithm

The DIRECT algorithm is a global optimization algorithm motivated by Lipschitzian optimization. DIRECT is deterministic and tailored to low-dimensional problems. It partitions the search space iteratively into increasingly smaller

rectangles which are each sampled on opposing vertices. DIRECT decides which rectangles are explored further, based on the value of those samples and the size of the rectangles. That way, rectangles are sampled either because they yield good values, or because they are large and therefore not yet sampled in detail. This leads to a balance between local refinement and global optimization.

The concrete implementation of DIRECT in *Simopticon* is a derivative of Adaptive Diagonal Curves and MrDIRECT which are both derivatives of the original DIRECT algorithm. For a more in-depth explanation of the implemented algorithm refer to the [german bachelor's thesis](#) that proposed it.

Monte Carlo Optimization

Monte Carlo methods are a simple class of random algorithms and therefore not deterministic. When applied to optimization problems, they show great performance despite their simplicity. Basically, the algorithm iteratively selects random values to be tested and evaluates them.

Random Neighbor Optimization

The RandomNeighbor [Optimizer](#) is an implementation of [random-restart stochastic hill climbing](#). It starts at a random point in the search space. In the next step, a new point is randomly chosen from the neighborhood of the current optimum. The next point is chosen completely random (i.e. independent of the optimum) with a predefined probability to ensure global search.

1.3.3 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

Module Index

3.1 Modules

Here is a list of all modules:

controller	19
evaluation	23
constant_headway	22
optimizer	23
direct	19
hyrect	24
montecarlo	21
randomneighbors	21
parameters	25
status	25
runner	26
plexo	22
utils	27

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Abortable	29
Controller	50
StubController	136
Optimizer	93
DirectOptimizer	57
MonteCarlo	85
RandomNeighbors	114
CmpPairVectorSharedParameterFunctionvalue	34
CmpPtrFunctionvalue	35
CmpSharedHyrect	36
CmpVectorSharedParameter	37
CommandLine	38
ConfigEditor	39
GrahamScan	72
HyRect	73
BaseRect	30
ChildRect	32
Levels	80
Multithreaded< Key, T, Compare, Allocator >	89
Multithreaded< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, Cmp↔ VectorSharedParameter >	89
SimulationRunner	119
PlexeSimulationRunner	107
Multithreaded< std::pair< std::filesystem::path, std::pair< std::string, unsigned int > >, bool >	89
PlexeSimulationRunner	107
Parameter	97
ContinuousParameter	48
DiscreteParameter	65
ParameterDefinition	102
ParameterNormalizer	105
PythonScript	112
ConstantHeadway	44
Status	123
Evaluation	70
ConstantHeadway	44
Optimizer	93
SimulationRunner	119
StatusBar	126

Controller::stepstate	129
StoppingCondition	131
DirectStoppingCondition	63
ThreadsafeQueue< Key >	140
ValueMap	142

Chapter 5

Class Index

5.1 Class List

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

Abortable	A simple interface for classes that encapsulate abortable processes	29
BaseRect	A class representing a HyRect without a parent rectangle	30
ChildRect	A class representing a HyRect that has a parent HyRect	32
CmpPairVectorSharedParameterFunctionvalue	This struct implements the comparison of two pairs of parameterCombination and function value	34
CmpPtrFunctionvalue	This struct implements the comparison of two pointers to function values	35
CmpSharedHyrect	This struct implements the comparison of two shared pointers to HyRect instances	36
CmpVectorSharedParameter	This struct implements the comparison of two vectors of Parameter references	37
CommandLine	A class containing functionality for executing commands on UNIX shell	38
ConfigEditor	A class capable of creating <code>.ini</code> files with certain options based on a complete <code>omnetpp.ini</code>	39
ConstantHeadway	A wrapper for the <code>constant_headway.py</code> script	44
ContinuousParameter	Implements a Parameter using continuous values in the form of floating point numbers	48
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	50
DirectOptimizer	A class capable of finding the minimum of a blackbox function using the DIRECT algorithm	57
DirectStoppingCondition	A class used for deciding whether the DIRECT should be stopped	63
DiscreteParameter	Implements a Parameter using discrete values	65
Evaluation	A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization	70
GrahamScan	A class providing functionality for finding the lower right convex hull of a set of points	72
HyRect	An abstract class representing a rectangular part of the search space	73

Levels	A providing functionality for the usage of different weightings between local and global search throughout the optimization using different levels	80
MonteCarlo	A class capable of finding the minimum of a blackbox function using the Monte Carlo algorithm	85
Multithreaded< Key, T, Compare, Allocator >	A class implementing concurrent execution of the same function for different arguments	89
Optimizer	A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs	93
Parameter	A class acting as the container of the value of a parameter defined by a ParameterDefinition .	97
ParameterDefinition	A class storing information on the properties of parameters that are being optimized	102
ParameterNormalizer	A class used for transforming parameters between the actual Parameter space and the unit hypercube used in DIRECT algorithm	105
PlexeSimulationRunner	A class capable of starting platooning simulations in the Plexe framework with given parameterCombinations	107
PythonScript	A class containing functionality for interfacing with the function of a Python module on creation	112
RandomNeighbors	A class capable of finding the minimum of a blackbox function using the Random Neighbors algorithm	114
SimulationRunner	A class capable of running simulations with certain parameterCombinations	119
Status	An interface defining functions for status updates on configuration and progress of a class . . .	123
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	126
Controller::stepstate	A struct keeping track of the currently running optimization step for StatusBar::updateStatus . .	129
StoppingCondition	A class used for deciding whether the optimization should be stopped	131
StubController	A class that mocks behaviour of Controller	136
ThreadsafeQueue< Key >	A container class of a queue that is safe for concurrent access of different threads	140
ValueMap	A container managing a map data structure that maps parameterCombinations to their respective found values	142

Chapter 6

File Index

6.1 File List

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

/home/runner/work/Simopticon/Simopticon/src/ComparisonFunctions.h	
In this file, comparison functions are defined which should be used across the whole framework	149
/home/runner/work/Simopticon/Simopticon/src/main.cpp	
In this file, the main function running the <i>Simopticon</i> framework is defined	161
/home/runner/work/Simopticon/Simopticon/src/Types.h	
In this file, types are defined which should be used across the whole framework	201
/home/runner/work/Simopticon/Simopticon/src/controller/Controller.cpp	
In this file, the implementation of the Controller class is defined	150
/home/runner/work/Simopticon/Simopticon/src/controller/Controller.h	
In this file, the header of the Controller class is defined	151
/home/runner/work/Simopticon/Simopticon/src/controller/StubController.cpp	
In this file, the implementation of the StubController class is defined	152
/home/runner/work/Simopticon/Simopticon/src/controller/StubController.h	
In this file, the header of the StubController class is defined	154
/home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.cpp	
In this file, the implementation of the ValueMap class is defined	155
/home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.h	
In this file, the header of the ValueMap class is defined	155
/home/runner/work/Simopticon/Simopticon/src/evaluation/Evaluation.cpp	
In this file, the implementation of the Evaluation class is defined	159
/home/runner/work/Simopticon/Simopticon/src/evaluation/Evaluation.h	
In this file, the header of the Evaluation class is defined	160
/home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/constant_headway.py	
In this file, Python functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap is defined	156
/home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.cpp	
In this file, the implementation of the ConstantHeadway class is defined	158
/home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.h	
In this file, the header of the ConstantHeadway class is defined	158
/home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.cpp	
In this file, the implementation of the Optimizer class is defined	180
/home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.h	
In this file, the header of the Optimizer class is defined	180
/home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.cpp	
In this file, the implementation of the StoppingCondition class is defined	183
/home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.h	
In this file, the header of the StoppingCondition class is defined	184
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectComparisonFunctions.h	
In this file, comparison functions are defined which are used in the direct module	163

/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ DirectOptimizer.cpp	
In this file, the implementation of the DirectOptimizer class is defined	164
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ DirectOptimizer.h	
In this file, the header of the DirectOptimizer class is defined	164
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ DirectStoppingCondition.cpp	
In this file, the implementation of the DirectStoppingCondition class is defined	165
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ DirectStoppingCondition.h	
In this file, the header of the StoppingCondition class is defined	166
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ DirectTypes.h	
In this file, types are defined which are used in the direct module	167
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ GrahamScan.cpp	
In this file, the implementation of the GrahamScan class is defined	169
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ GrahamScan.h	
In this file, the header of the GrahamScan class is defined	169
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ Levels.cpp	
In this file, the implementation of the Levels class is defined	175
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ Levels.h	
In this file, the header of the Levels class is defined	175
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ ParameterNormalizer.cpp	
In this file, the implementation of the ParameterNormalizer class is defined	177
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ ParameterNormalizer.h	
In this file, the header of the ParameterNormalizer class is defined	177
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ BaseRect.cpp	
In this file, the implementation of the BaseRect class is defined	170
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ BaseRect.h	
In this file, the header of the BaseRect class is defined	171
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ ChildRect.cpp	
In this file, the implementation of the ChildRect class is defined	172
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ ChildRect.h	
In this file, the header of the ChildRect class is defined	172
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ HyRect.cpp	
In this file, the implementation of the HyRect class is defined	173
/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/ HyRect.h	
In this file, the header of the HyRect class is defined	174
/home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/ MonteCarlo.cpp	
In this file, the implementation of the MonteCarlo optimizer class is defined	179
/home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/ MonteCarlo.h	
In this file, the header of the MonteCarlo class is defined	179
/home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors/ RandomNeighbors.cpp	
In this file, the implementation of the RandomNeighbors optimizer class is defined	181
/home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors/ RandomNeighbors.h	
In this file, the header of the RandomNeighbors optimizer class is defined	182
/home/runner/work/Simopticon/Simopticon/src/parameters/ ContinuousParameter.cpp	
In this file, the implementation of the ContinuousParameter class is defined	185
/home/runner/work/Simopticon/Simopticon/src/parameters/ ContinuousParameter.h	
In this file, the header of the ContinuousParameter class is defined	186
/home/runner/work/Simopticon/Simopticon/src/parameters/ DiscreteParameter.cpp	
In this file, the implementation of the DiscreteParameter class is defined	187
/home/runner/work/Simopticon/Simopticon/src/parameters/ DiscreteParameter.h	
In this file, the header of the DiscreteParameter class is defined	188
/home/runner/work/Simopticon/Simopticon/src/parameters/ Parameter.cpp	
In this file, the implementation of the Parameter class is defined	189
/home/runner/work/Simopticon/Simopticon/src/parameters/ Parameter.h	
In this file, the header of the Parameter class is defined	189
/home/runner/work/Simopticon/Simopticon/src/parameters/ ParameterDefinition.cpp	
In this file, the implementation of the ParameterDefinition class is defined	190
/home/runner/work/Simopticon/Simopticon/src/parameters/ ParameterDefinition.h	
In this file, the header of the ParameterDefinition class is defined	191

/home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.cpp	
In this file, the implementation of the SimulationRunner class is defined	196
/home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.h	
In this file, the header of the SimulationRunner class is defined	196
/home/runner/work/Simopticon/Simopticon/src/runner/plex/ConfigEditor.cpp	
In this file, the implementation of the ConfigEditor class is defined	192
/home/runner/work/Simopticon/Simopticon/src/runner/plex/ConfigEditor.h	
In this file, the header of the ConfigEditor class is defined	193
/home/runner/work/Simopticon/Simopticon/src/runner/plex/PlexSimulationRunner.cpp	
In this file, the implementation of the PlexSimulationRunner class is defined	194
/home/runner/work/Simopticon/Simopticon/src/runner/plex/PlexSimulationRunner.h	
In this file, the header of the PlexSimulationRunner class is defined	195
/home/runner/work/Simopticon/Simopticon/src/status/Status.cpp	
In this file, the implementation of the Status class is defined	197
/home/runner/work/Simopticon/Simopticon/src/status/Status.h	
In this file, the header of the Status class is defined	198
/home/runner/work/Simopticon/Simopticon/src/status/StatusBar.cpp	
In this file, the implementation of the StatusBar class is defined	199
/home/runner/work/Simopticon/Simopticon/src/status/StatusBar.h	
In this file, the header of the StatusBar class is defined	199
/home/runner/work/Simopticon/Simopticon/src/utis/Abortable.cpp	
In this file, the implementation of the Abortable class is defined	202
/home/runner/work/Simopticon/Simopticon/src/utis/Abortable.h	
In this file, the header of the Abortable class is defined	203
/home/runner/work/Simopticon/Simopticon/src/utis/CommandLine.cpp	
In this file, the implementation of the CommandLine class is defined	203
/home/runner/work/Simopticon/Simopticon/src/utis/CommandLine.h	
In this file, the header of the CommandLine class is defined	204
/home/runner/work/Simopticon/Simopticon/src/utis/Multithreaded.h	
In this file, the header of the Multithreaded class is defined	205
/home/runner/work/Simopticon/Simopticon/src/utis/Multithreaded.tpp	
In this file, the implementation of the Multithreaded class is defined	205
/home/runner/work/Simopticon/Simopticon/src/utis/PythonScript.cpp	
In this file, the implementation of the PythonScript class is defined	206
/home/runner/work/Simopticon/Simopticon/src/utis/PythonScript.h	
In this file, the header of the PythonScript class is defined	207
/home/runner/work/Simopticon/Simopticon/src/utis/ThreadSafeQueue.h	
In this file, the header of the ThreadSafeQueue class is defined	208
/home/runner/work/Simopticon/Simopticon/src/utis/ThreadSafeQueue.tpp	
In this file, the implementation of the ThreadSafeQueue class is defined	209

Chapter 7

Module Documentation

7.1 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 [parameterCombinations](#) to their respective found values.

Variables

- struct [Controller::stepstate](#) [Controller::stepState](#)
An object keeping track of the current optimization step.

7.1.1 Detailed Description

This module provides classes coordinating the optimization process independently from the actual implementation of [Optimizer](#), [SimulationRunner](#) and [Evaluation](#).

7.1.2 Variable Documentation

7.1.2.1 stepState

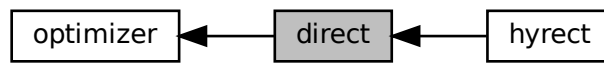
```
struct Controller::stepstate Controller::stepState [protected]
```

An object keeping track of the current optimization step.

7.2 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 used 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 [DirectStoppingCondition](#)

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.

7.2.1 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.](#)

Author

Per Natzschka

7.2.2 Enumeration Type Documentation

7.2.2.1 level

enum `level` : unsigned char

An enum representing the sequence of local levels.

Definition at line 24 of file Levels.h.

7.3 montecarlo

This is a simple implementation of Monte Carlo optimization.

Collaboration diagram for montecarlo:



Classes

- class `MonteCarlo`

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

7.3.1 Detailed Description

This is a simple implementation of Monte Carlo optimization.

`Parameter` values are chosen by random in given bounds.

7.4 randomneighbors

This is a simple implementation of Random Neighbors optimization.

Collaboration diagram for randomneighbors:



Classes

- class `RandomNeighbors`

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

7.4.1 Detailed Description

This is a simple implementation of Random Neighbors optimization.

[Parameter](#) values are chosen by random in given bounds. Besides global search, it is searched more often in the neighbourhood of the current optimum.

7.5 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 parameterCombinations.
- class [ConfigEditor](#)
A class capable of creating `.ini` files with certain options based on a complete `omnetpp.ini`.

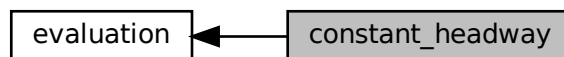
7.5.1 Detailed Description

This module extends [SimulationRunner](#) to interface with the Plexe framework to enable the optimization of platooning controllers.

7.6 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](#)
In this file, Python functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap is defined.

Classes

- class [ConstantHeadway](#)
A wrapper for the `constant_headway.py` script.

7.6.1 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.

Author

Per Natzschka

7.7 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.

7.7.1 Detailed Description

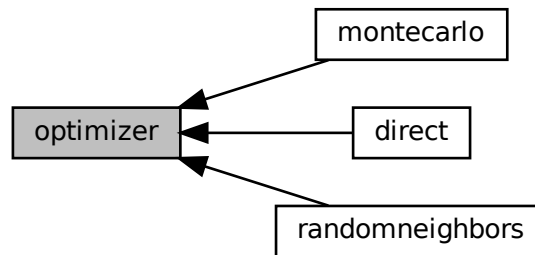
This module contains components capable of evaluating the performance of simulations by rating simulation data with a number value.

Implementations must extend [Evaluation](#).

7.8 optimizer

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

Collaboration diagram for optimizer:



Modules

- [direct](#)
This module extends [Optimizer](#) to use a variant of the DIRECT algorithm by [Jones et al.](#)
- [montecarlo](#)
This is a simple implementation of Monte Carlo optimization.
- [randomneighbors](#)
This is a simple implementation of Random Neighbors optimization.

Classes

- class [Optimizer](#)
A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.
- class [StoppingCondition](#)
A class used for deciding whether the optimization should be stopped.

7.8.1 Detailed Description

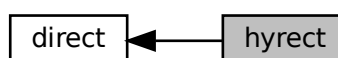
This module contains components capable of finding the minimum of a function only defined through argument-value pairs.

Implementations must extend [Optimizer](#).

7.9 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](#).

7.9.1 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](#)).

7.9.2 Enumeration Type Documentation

7.9.2.1 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 35 of file [HyRect.h](#).

7.10 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.

7.10.1 Detailed Description

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

7.11 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.

7.11.1 Detailed Description

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

7.11.2 Enumeration Type Documentation

7.11.2.1 step

```
enum step : char
```

An Enum defining the steps, an optimization process cycles through.
Definition at line 30 of file StatusBar.h.

7.12 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 parameterCombinations.

7.12.1 Detailed Description

This module contains components capable of automatically running simulations with certain parameter↵
Combinations.

Implementations must extend [SimulationRunner](#).

7.13 utils

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

Files

- file [main.cpp](#)
In this file, the main function running the Simopticon framework is defined.
- 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](#)
In this file, the main function running the Simopticon framework is defined.

Classes

- 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 [CommandLine](#)
A class containing functionality for executing commands on UNIX shell.
- class [PythonScript](#)
A class containing functionality for interfacing with the function of a Python module on creation.
- class [ThreadsafeQueue< Key >](#)
A container class of a queue that is safe for concurrent access of different threads.

7.13.1 Detailed Description

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

Chapter 8

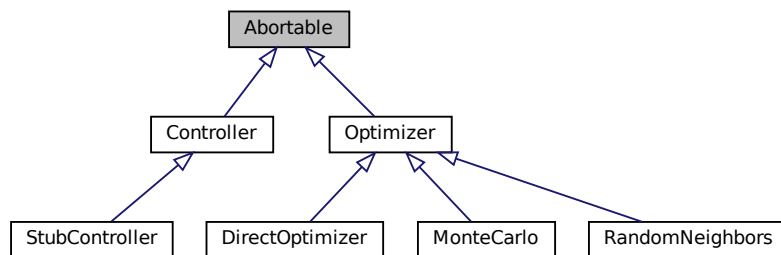
Class Documentation

8.1 Abortable Class Reference

A simple interface for classes that encapsulate abortable processes.

```
#include "Abortable.h"
```

Inheritance diagram for Abortable:



Collaboration 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.

8.1.1 Detailed Description

A simple interface for classes that encapsulate abortable processes.

Author

Per Natzschka

Definition at line 14 of file Abortable.h.

8.1.2 Member Function Documentation

8.1.2.1 abort()

```
void Abortable::abort ( ) [virtual]
```

Sets `aborted` to `true`.

Reimplemented in [Controller](#).

Definition at line 9 of file Abortable.cpp.

References `aborted`.

Referenced by `Controller::abort()`.

8.1.3 Member Data Documentation

8.1.3.1 aborted

```
bool Abortable::aborted = false [protected]
```

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

`abort` has been called.

Definition at line 19 of file Abortable.h.

Referenced by `abort()`.

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

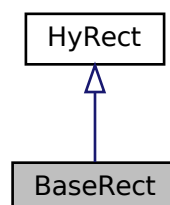
- [/home/runner/work/Simopticon/Simopticon/src/utils/Abortable.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/utils/Abortable.cpp](#)

8.2 BaseRect Class Reference

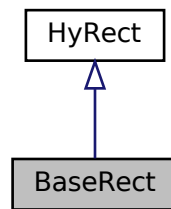
A class representing a [HyRect](#) without a parent rectangle.

```
#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.
- `std::array< std::vector< dirCoordinate >, 2 > getSamplingVertices ()` override
Returns the coordinates of two opposite corner points of the rectangle.

Additional Inherited Members

8.2.1 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.

Author

Per Natzschka

Definition at line 17 of file BaseRect.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 BaseRect()

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

Creates a [BaseRect](#) representing a hypercube with the given dimensionality.

Parameters

<i>D</i>	Number of dimensions of the search space.
----------	---

Definition at line 9 of file BaseRect.cpp.

References [HyRect::HyRect\(\)](#).

8.2.3 Member Function Documentation

8.2.3.1 getSamplingVertices()

```
std::array< std::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 12 of file [BaseRect.cpp](#).

References [HyRect::D](#).

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

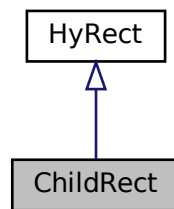
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/BaseRect.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/BaseRect.cpp](#)

8.3 ChildRect Class Reference

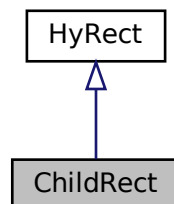
A class representing a [HyRect](#) that has a parent [HyRect](#).

```
#include "ChildRect.h"
```

Inheritance diagram for [ChildRect](#):



Collaboration diagram for [ChildRect](#):



Public Member Functions

- [ChildRect](#) (position *pos*, std::shared_ptr< [HyRect](#) > *parent*)
Creates a *ChildRect* with the given relative position and parent rectangle.
- std::array< std::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

- std::shared_ptr< [HyRect](#) > *parent*
Reference to the parent rectangle.

Additional Inherited Members

8.3.1 Detailed Description

A class representing a [HyRect](#) that has a parent [HyRect](#).
Used for all [HyRect](#) where depth $t > 0$.

Author

Per Natzschka

Definition at line 17 of file [ChildRect.h](#).

8.3.2 Constructor & Destructor Documentation

8.3.2.1 ChildRect()

```
ChildRect::ChildRect (
    position pos,
    std::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 11 of file [ChildRect.cpp](#).

References [HyRect::HyRect\(\)](#), [HyRect::getD\(\)](#), [HyRect::getDepth\(\)](#), and [parent](#).

Referenced by [HyRect::divide\(\)](#).

8.3.3 Member Function Documentation

8.3.3.1 getSamplingVertices()

```
std::array< std::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 16 of file `ChildRect.cpp`.

References [HyRect::getSamplingVertices\(\)](#), [HyRect::getSplitDim\(\)](#), [parent](#), and [HyRect::pos](#).

8.3.3.2 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>	HyRect to be compared.
-------------	--

Returns

A boolean defining if the [HyRect](#) objects have the same position in the partition tree.

Reimplemented from [HyRect](#).

Definition at line 34 of file `ChildRect.cpp`.

References [HyRect::getPos\(\)](#), [parent](#), and [HyRect::pos](#).

8.3.4 Member Data Documentation**8.3.4.1 parent**

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

Reference to the parent rectangle.

Used for recursive calculation of [getSamplingVertices](#).

Definition at line 22 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`

8.4 CmpPairVectorSharedParameterFunctionvalue Struct Reference

This struct implements the comparison of two pairs of `parameterCombination` and `functionvalue`.

```
#include "ComparisonFunctions.h"
```

Collaboration diagram for `CmpPairVectorSharedParameterFunctionvalue`:

CmpPairVectorSharedParameter Functionvalue

Public Member Functions

- bool `operator()` (const std::pair< `parameterCombination`, `functionValue` > &a, const std::pair< `parameterCombination`, `functionValue` > &b) const
Compares two pairs of parameterCombination and function value.

8.4.1 Detailed Description

This struct implements the comparison of two pairs of parameterCombination and function value. Definition at line 56 of file ComparisonFunctions.h.

8.4.2 Member Function Documentation

8.4.2.1 operator()

```
bool CmpPairVectorSharedParameterFunctionvalue::operator() (
    const std::pair< parameterCombination, functionValue > & a,
    const std::pair< parameterCombination, functionValue > & b ) const [inline]
```

Compares two pairs of parameterCombination 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 parameterCombinations are compared.

Definition at line 63 of file ComparisonFunctions.h.

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

- [/home/runner/work/Simopticon/Simopticon/src/ComparisonFunctions.h](#)

8.5 CmpPtrFunctionvalue Struct Reference

This struct implements the comparison of two pointers to function values.

```
#include "ComparisonFunctions.h"
```

Collaboration diagram for CmpPtrFunctionvalue:

CmpPtrFunctionvalue

Public Member Functions

- bool `operator()` (const `functionValue` *a, const `functionValue` *b) const
Compares two pointers to function values.

8.5.1 Detailed Description

This struct implements the comparison of two pointers to function values.
Definition at line 43 of file ComparisonFunctions.h.

8.5.2 Member Function Documentation

8.5.2.1 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 50 of file ComparisonFunctions.h.

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

- /home/runner/work/Simopticon/Simopticon/src/[ComparisonFunctions.h](#)

8.6 CmpSharedHyrect Struct Reference

This struct implements the comparison of two shared pointers to [HyRect](#) instances.

```
#include "DirectComparisonFunctions.h"
```

Collaboration diagram for CmpSharedHyrect:

CmpSharedHyrect

Public Member Functions

- bool [operator\(\)](#) (const std::shared_ptr< [HyRect](#) > &a, const std::shared_ptr< [HyRect](#) > &b) const
Compares two shared pointers to [HyRect](#) instances.

8.6.1 Detailed Description

This struct implements the comparison of two shared pointers to [HyRect](#) instances.

Definition at line 18 of file DirectComparisonFunctions.h.

8.6.2 Member Function Documentation

8.6.2.1 operator()

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

Compares two shared pointers to [HyRect](#) instances.

Parameters

<i>a</i>	First pointer to a HyRect .
<i>b</i>	Second pointer to a HyRect .

Returns

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

Definition at line 25 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](#)

8.7 CmpVectorSharedParameter Struct Reference

This struct implements the comparison of two vectors of [Parameter](#) references.

```
#include "ComparisonFunctions.h"
```

Collaboration diagram for [CmpVectorSharedParameter](#):

CmpVectorSharedParameter

Public Member Functions

- bool [operator\(\)](#) ([parameterCombination](#) a, [parameterCombination](#) b) const
Compares two vectors of [Parameter](#) references.

8.7.1 Detailed Description

This struct implements the comparison of two vectors of [Parameter](#) references.

Definition at line 20 of file [ComparisonFunctions.h](#).

8.7.2 Member Function Documentation

8.7.2.1 operator()

```
bool CmpVectorSharedParameter::operator() (
    parameterCombination a,
    parameterCombination 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 27 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>

8.8 CommandLine Class Reference

A class containing functionality for executing commands on UNIX shell.

```
#include "CommandLine.h"
```

Collaboration diagram for CommandLine:

CommandLine

Static Public Member Functions

- `static std::unique_ptr< std::string > exec (std::string cmd)`
Executes the given command in UNIX shell and returns the output (both stderr and stdout merged).

8.8.1 Detailed Description

A class containing functionality for executing commands on UNIX shell.

Author

Per Natzschka

Definition at line 27 of file CommandLine.h.

8.8.2 Member Function Documentation

8.8.2.1 exec()

```
std::unique_ptr< std::string > CommandLine::exec (
    std::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 (stderr and stdout merged).

Definition at line 13 of file CommandLine.cpp.

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

- /home/runner/work/Simopticon/Simopticon/src/utills/CommandLine.h
- /home/runner/work/Simopticon/Simopticon/src/utills/CommandLine.cpp

8.9 ConfigEditor Class Reference

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

```
#include "ConfigEditor.h"
```

Collaboration diagram for ConfigEditor:

```

classDiagram
    class ConfigEditor
  
```

Public Member Functions

- [ConfigEditor](#) (std::filesystem::path directory, nlohmann::json controller)

Creates a [ConfigEditor](#) that creates config files in the given directory for simulation of the given controller.
- void [createConfig](#) (const [parameterCombination](#) ¶ms, 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 std::filesystem::path & [getDir](#) () const

Returns the directory of the Plexe configuration.
- std::filesystem::path [getConfigPath](#) (size_t runId) const

Returns the path to the created config for the parameterCombination with the given number.
- std::filesystem::path [getResultPath](#) (size_t runId) const

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

Private Member Functions

- void [setResultFiles](#) (std::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](#) (std::string &file, std::string option, const std::string &value)

Replaces the value of the given key with the given new value in the given string.
- static void [replaceOption](#) (std::string &file, std::string option, long value)

Replaces the value of the given key with the given new value in the given string.
- static std::string [getControllerOption](#) (std::string &file)

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

Private Attributes

- `const std::filesystem::path DIR`
Path to a directory containing a complete configuration of Plexe.
- `const std::filesystem::path CONFIG`
Path to the `omnetpp.ini` file in `DIR`.
- `const std::filesystem::path RESULTS`
Path to the `optResults` directory in `DIR` where the simulation result files are generated.
- `const nlohmann::json CONTROLLER`
Configuration of the controller to be simulated.

8.9.1 Detailed Description

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

Author

Per Natzschka

Definition at line 25 of file `ConfigEditor.h`.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 ConfigEditor()

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

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

Parameters

<code>directory</code>	A path to the directory containing a Plexe configuration.
<code>controller</code>	A json object configuring the controller to be simulated.

Definition at line 12 of file `ConfigEditor.cpp`.

References `ConfigEditor()`.

Referenced by `ConfigEditor()`.

8.9.3 Member Function Documentation

8.9.3.1 createConfig()

```
void ConfigEditor::createConfig (
    const parameterCombination & 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

<code>params</code>	The parameterCombination to be simulated.
---------------------	---

Parameters

<i>runNumber</i>	An unique number of the simulated parameterCombination.
<i>repeat</i>	Number of repetitions to be simulated.

Definition at line 17 of file ConfigEditor.cpp.
References setResultFiles().

8.9.3.2 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 96 of file ConfigEditor.cpp.
References getConfigPath().

8.9.3.3 getConfigPath()

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

Returns the path to the created config for the parameterCombination with the given number.

Parameters

<i>runId</i>	Number of the parameterCombination.
--------------	-------------------------------------

Returns

A path to the config for the given runId.

Definition at line 86 of file ConfigEditor.cpp.
Referenced by deleteConfig().

8.9.3.4 getControllerOption()

```
std::string ConfigEditor::getControllerOption (
    std::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 <code>.ini</code> file.
-------------	--

Returns

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

Definition at line 75 of file ConfigEditor.cpp.

8.9.3.5 getDir()

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

Returns the directory of the Plexe configuration.

Returns

The path stored in [DIR](#)

Definition at line 100 of file ConfigEditor.cpp.

8.9.3.6 getResultPath()

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

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

Parameters

<i>runId</i>	Number of the parameterCombination.
--------------	-------------------------------------

Returns

A path to the result files for the given runId.

Definition at line 91 of file ConfigEditor.cpp.

8.9.3.7 replaceOption() [1/2]

```
void ConfigEditor::replaceOption (
    std::string & file,
    std::string option,
    const std::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 48 of file ConfigEditor.cpp.

8.9.3.8 replaceOption() [2/2]

```
void ConfigEditor::replaceOption (
    std::string & file,
    std::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(std::string &, std::string, const std::string &)`.

Parameters

<i>file</i>	A string containing the contents of an <code>.ini</code> 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 62 of file `ConfigEditor.cpp`.

8.9.3.9 setResultFiles()

```
void ConfigEditor::setResultFiles (
    std::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 <code>.ini</code> file.
<i>runNumber</i>	The unique number of the parameterCombination.

Definition at line 66 of file `ConfigEditor.cpp`.
 Referenced by `createConfig()`.

8.9.4 Member Data Documentation

8.9.4.1 CONFIG

```
const std::filesystem::path ConfigEditor::CONFIG [private]
```

Path to the `omnetpp.ini` file in **DIR**.

Definition at line 35 of file `ConfigEditor.h`.

8.9.4.2 CONTROLLER

```
const nlohmann::json ConfigEditor::CONTROLLER [private]
```

Configuration of the controller to be simulated.

Can be set in config.

Definition at line 44 of file `ConfigEditor.h`.

8.9.4.3 DIR

```
const std::filesystem::path ConfigEditor::DIR [private]
```

Path to a directory containing a complete configuration of Plexe.

Can be set in config.

Definition at line 31 of file `ConfigEditor.h`.

8.9.4.4 RESULTS

```
const std::filesystem::path ConfigEditor::RESULTS [private]
```

Path to the `optResults` directory in `DIR` where the simulation result files are generated.
Definition at line 39 of file `ConfigEditor.h`.

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

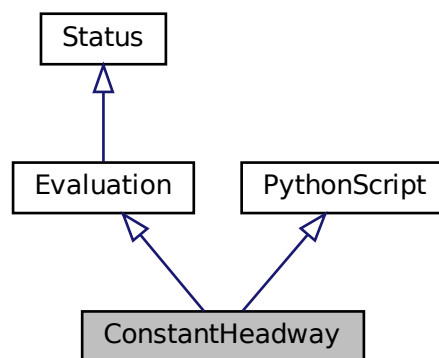
- [/home/runner/work/Simopticon/Simopticon/src/runner/plexo/ConfigEditor.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/runner/plexo/ConfigEditor.cpp](#)

8.10 ConstantHeadway Class Reference

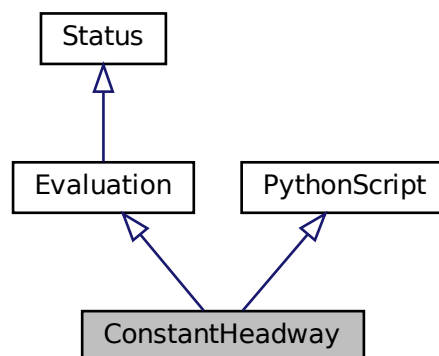
A wrapper for the `constant_headway.py` script.

```
#include "ConstantHeadway.h"
```

Inheritance diagram for `ConstantHeadway`:



Collaboration diagram for `ConstantHeadway`:



Public Member Functions

- `ConstantHeadway` (unsigned int nrThreads, const std::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](#) (std::filesystem::path path, std::set< [runId](#) > experimentIds) override
Returns a value to the results of a single simulation run.
- std::map< std::pair< std::filesystem::path, std::set< [runId](#) > >, [functionValue](#) > [processOutput](#) (const std::set< std::pair< std::filesystem::path, std::set< [runId](#) >>> &experimentResults) override
Returns values to the results of multiple simulation runs.
- std::string [getName](#) () override
Returns a string representing the name of the implementing component in natural language.
- std::string [getStatus](#) () override
Returns a string representing the current state of the implementing component.
- std::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

8.10.1 Detailed Description

A wrapper for the [constant_headway.py](#) script.
Definition at line 36 of file ConstantHeadway.h.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 ConstantHeadway()

```
ConstantHeadway::ConstantHeadway (
    unsigned int nrThreads,
    const std::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 constant_headway.py script.

Definition at line 11 of file ConstantHeadway.cpp.
References PythonScript::PythonScript(), and NR_THREADS.

8.10.3 Member Function Documentation

8.10.3.1 getName()

```
std::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 76 of file ConstantHeadway.cpp.

8.10.3.2 getStatus()

```
std::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 80 of file ConstantHeadway.cpp.

References NR_THREADS.

8.10.3.3 getStatusBar()

```
std::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 84 of file ConstantHeadway.cpp.

References NR_THREADS, and usedThreads.

8.10.3.4 processOutput() [1/2]

```
std::map< std::pair< std::filesystem::path, std::set< runId > >, functionValue > Constant↔
Headway::processOutput (
    const std::set< std::pair< std::filesystem::path, std::set< runId >>> & experiment↔
Results ) [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 16 of file ConstantHeadway.cpp.
References usedThreads.

8.10.3.5 processOutput() [2/2]

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

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

Basically calls `processOutput(const std::set<std::pair<std::filesystem::path, std::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 61 of file ConstantHeadway.cpp.

8.10.3.6 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 66 of file ConstantHeadway.cpp.

8.10.4 Member Data Documentation

8.10.4.1 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 42 of file ConstantHeadway.h.

Referenced by `ConstantHeadway()`, `getStatus()`, and `getStatusBar()`.

8.10.4.2 usedThreads

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

Number of threads currently used for concurrent evaluation.

Used in [getStatusBar](#).

Definition at line 47 of file ConstantHeadway.h.

Referenced by `getStatusBar()`, and `processOutput()`.

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

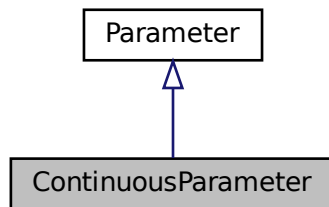
- /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.h
- /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.cpp

8.11 ContinuousParameter Class Reference

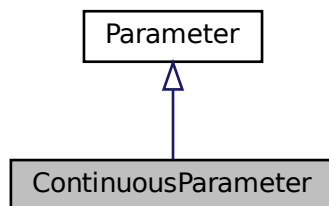
Implements a [Parameter](#) using continuous values in the form of floating point numbers.

```
#include "ContinuousParameter.h"
```

Inheritance diagram for ContinuousParameter:



Collaboration diagram for ContinuousParameter:



Public Member Functions

- [ContinuousParameter](#) (std::shared_ptr< [ParameterDefinition](#) > def, [coordinate](#) value)
Creates a [ContinuousParameter](#) with the given [ParameterDefinition](#) and value.
- [ContinuousParameter](#) (std::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](#) newVal) override
Sets the value of [val](#) to the given value.

Private Attributes

- [coordinate](#) `val`

Value of the [ContinuousParameter](#).

8.11.1 Detailed Description

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

Author

Per Natzschka

Definition at line 16 of file [ContinuousParameter.h](#).

8.11.2 Constructor & Destructor Documentation

8.11.2.1 ContinuousParameter() [1/2]

```
ContinuousParameter::ContinuousParameter (
    std::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>	ParameterDefinition of the Parameter .
<i>value</i>	Initial value of the Parameter .

Definition at line 12 of file [ContinuousParameter.cpp](#).

References [Parameter::Parameter\(\)](#), [Parameter::getMax\(\)](#), [Parameter::getMin\(\)](#), and `val`.

Referenced by [ContinuousParameter\(\)](#), and [ParameterNormalizer::denormalize\(\)](#).

8.11.2.2 ContinuousParameter() [2/2]

```
ContinuousParameter::ContinuousParameter (
    std::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>	ParameterDefinition of the Parameter .
------------	--

Definition at line 19 of file [ContinuousParameter.cpp](#).

References [ContinuousParameter\(\)](#), [Parameter::getMax\(\)](#), and [Parameter::getMin\(\)](#).

8.11.3 Member Function Documentation

8.11.3.1 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 23 of file `ContinuousParameter.cpp`.

References [val](#).

8.11.3.2 `setVal()`

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

Sets the value of [val](#) to the given value.

Checks if given value is in bounds set by the [ParameterDefinition](#).

Parameters

<code>newVal</code>	Value to set the ContinuousParameter to.
---------------------	--

Implements [Parameter](#).

Definition at line 27 of file `ContinuousParameter.cpp`.

References [Parameter::getMax\(\)](#), [Parameter::getMin\(\)](#), and [val](#).

8.11.4 Member Data Documentation

8.11.4.1 `val`

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

Value of the [ContinuousParameter](#).

Definition at line 21 of file `ContinuousParameter.h`.

Referenced by [ContinuousParameter\(\)](#), [getVal\(\)](#), and [setVal\(\)](#).

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

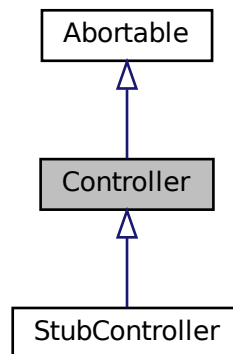
- [/home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousParameter.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousParameter.cpp](#)

8.12 Controller Class Reference

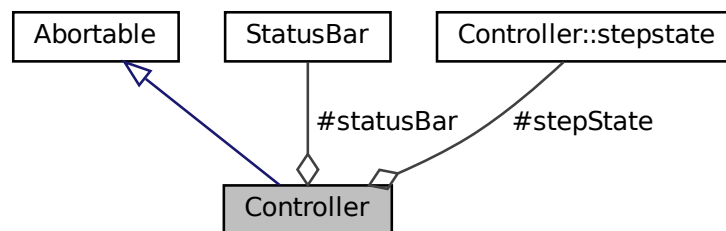
A class responsible for communication between [Optimizer](#), [SimulationRunner](#) and [Evaluation](#) and also user interaction such as tracking results, updating [StatusBar](#) and handling interrupts by the user via [Abortable](#).

```
#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 std::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](#).
- std::map< [parameterCombination](#), [functionValue](#) > [requestValues](#) (const std::list< [parameterCombination](#) > ¶ms)
Searches [valueMap](#) for results to given parameterCombinations.
- [ValueMap](#) & [getValueMap](#) ()
Returns [valueMap](#).
- void [abort](#) () override
Aborts [optimizer](#) using [Optimizer::abort](#).

Protected Attributes

- [StatusBar](#) `statusBar`
StatusBar object used for output.
- `std::unique_ptr< Optimizer > optimizer`
Optimizer defining an optimization strategy.
- `std::unique_ptr< SimulationRunner > runner`
SimulationRunner able to run simulations with certain *parameterCombinations*.
- `std::unique_ptr< Evaluation > evaluation`
Evaluation capable of evaluating data produced by *runner*.
- `std::unique_ptr< ValueMap > valueMap`
ValueMap containing all values gathered by simulating and evaluating certain *parameterCombinations*.
- struct `Controller::stepstate stepState`
An object keeping track of the current optimization step.

Private Member Functions

- virtual `std::map< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, CmpVectorSharedParameter > runSimulations` (const `std::set< parameterCombination, CmpVectorSharedParameter > &runs`)
Calls the *runner* to run simulations for the given *parameterCombinations*.
- virtual `std::map< parameterCombination, functionValue, CmpVectorSharedParameter > evaluate` (const `std::map< parameterCombination, std::pair< std::filesystem::path, std::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.
- void `saveValues ()`
Prints all evaluated *parameterCombinations* and their respective values to *results/values.csv*.
- virtual void `updateStatus ()`
Updates the *statusBar* using *StatusBar::updateStatus*.

Private Attributes

- bool `keepFiles`
Defines if result files of best simulations are kept after optimization.
- bool `printValues`
Defines if all found values should be recorded in a *.csv* file after optimization has finished.
- `std::map< parameterCombination, std::filesystem::path > topResults`
Saves the best *n* *parameterCombinations* and the corresponding path to the result files, if *keepFiles* is true.
- `std::chrono::milliseconds statusInterval = std::chrono::milliseconds(0)`
Interval of updates of *StatusBar* using *updateStatus* in concurrent status thread.

8.12.1 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](#).

Author

Per Natzschka

Definition at line 44 of file `Controller.h`.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 Controller()

```
Controller::Controller (
    const std::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

<i>configPath</i>	Path to the main config. Chosen by first command line argument.
<i>isStub</i>	Defines whether the constructor was called by constructor of StubController .

Definition at line 40 of file `Controller.cpp`.
References `statusInterval`, and `valueMap`.
Referenced by `StubController::StubController()`.

8.12.3 Member Function Documentation

8.12.3.1 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 275 of file `Controller.cpp`.

References `Abortable::abort()`, and `optimizer`.

8.12.3.2 evaluate()

```
std::map< parameterCombination, functionValue, CmpVectorSharedParameter > Controller::evaluate (
```

```
    const std::map< parameterCombination, std::pair< std::filesystem::path, std::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 parameterCombinations that must be evaluated to their respective file paths of simulation results and runIds.
--------------------------	--

Returns

A map which maps the given parameterCombinations to their respective functionValue.

Reimplemented in [StubController](#).

Definition at line 208 of file `Controller.cpp`.

References `updateStatus()`.

8.12.3.3 getValueMap()

`ValueMap & Controller::getValueMap ()`

Returns [valueMap](#).

Returns

A [ValueMap](#) object.

Definition at line 173 of file `Controller.cpp`.

References [valueMap](#).

Referenced by `Optimizer::getValueMap()`.

8.12.3.4 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 224 of file `Controller.cpp`.

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

Referenced by `requestValues()`.

8.12.3.5 requestValues()

`std::map< parameterCombination, functionValue > Controller::requestValues (
 const std::list< parameterCombination > & params)`

Searches [valueMap](#) for results to given [parameterCombinations](#).

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 parameterCombinations to be evaluated.
---------------	---

Returns

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

Definition at line 139 of file `Controller.cpp`.

References `Controller::stepstate::next()`, `removeOldResultfiles()`, and `updateStatus()`.

Referenced by `Optimizer::requestValues()`.

8.12.3.6 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 177 of file `Controller.cpp`.

References `Controller::stepstate::next()`, `optimizer`, `printValues`, `Optimizer::runOptimization()`, `saveValues()`, `statusInterval`, and `updateStatus()`.

8.12.3.7 runSimulations()

`std::map< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, CmpVectorSharedParamete
 > Controller::runSimulations (`

```
const std::set< parameterCombination, CmpVectorSharedParameter > & runs ) [private],
[virtual]
```

Calls the [runner](#) to run simulations for the given parameterCombinations.
Updates [statusBar](#) before and after execution of simulations.

Parameters

<code>runs</code>	A set of parameterCombinations to be executed.
-------------------	--

Returns

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

Reimplemented in [StubController](#).

Definition at line 201 of file Controller.cpp.

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

8.12.3.8 saveValues()

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

Prints all evaluated parameterCombinations and their respective values to `results/values.csv`.

Definition at line 242 of file Controller.cpp.

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

8.12.3.9 updateStatus()

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

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

Reimplemented in [StubController](#).

Definition at line 267 of file Controller.cpp.

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

Referenced by [evaluate\(\)](#), [requestValues\(\)](#), [run\(\)](#), and [runSimulations\(\)](#).

8.12.4 Member Data Documentation

8.12.4.1 evaluation

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

[Evaluation](#) capable of evaluating data produced by [runner](#).

Definition at line 114 of file Controller.h.

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

8.12.4.2 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 49 of file Controller.h.

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

8.12.4.3 optimizer

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

[Optimizer](#) defining an optimization strategy.

Definition at line 106 of file Controller.h.

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

8.12.4.4 printValues

`bool Controller::printValues [private]`

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

Can be set in main config.

Definition at line 54 of file Controller.h.

Referenced by `run()`.

8.12.4.5 runner

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

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

Definition at line 110 of file Controller.h.

Referenced by `updateStatus()`.

8.12.4.6 statusBar

`StatusBar Controller::statusBar [protected]`

[StatusBar](#) object used for output.

Definition at line 102 of file Controller.h.

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

8.12.4.7 statusInterval

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

Interval of updates of [StatusBar](#) using `updateStatus` in concurrent status thread.

Definition at line 62 of file Controller.h.

Referenced by `Controller()`, and `run()`.

8.12.4.8 topResults

`std::map<parameterCombination, std::filesystem::path> Controller::topResults [private]`

Saves the best n parameterCombinations and the corresponding path to the result files, if `keepFiles` is true.

n can be set in main config.

Definition at line 58 of file Controller.h.

Referenced by `removeOldResultfiles()`.

8.12.4.9 valueMap

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

[ValueMap](#) containing all values gathered by simulating and evaluating certain parameterCombinations.

Definition at line 118 of file Controller.h.

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

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

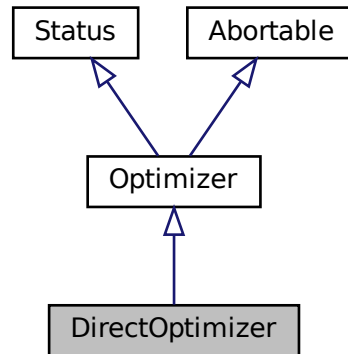
- `/home/runner/work/Simopticon/Simopticon/src/controller/Controller.h`
- `/home/runner/work/Simopticon/Simopticon/src/controller/Controller.cpp`

8.13 DirectOptimizer Class Reference

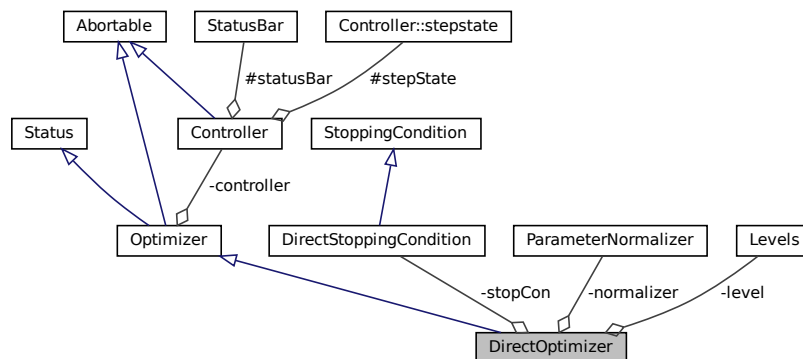
A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.

```
#include "DirectOptimizer.h"
```

Inheritance diagram for DirectOptimizer:



Collaboration diagram for DirectOptimizer:



Public Member Functions

- `DirectOptimizer (Controller &ctrl, const std::list< std::shared_ptr< ParameterDefinition >> ¶ms, DirectStoppingCondition con, bool trackProgress)`
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.
- `std::string getName ()` override
Returns a string representing the name of the implementing component in natural language.
- `std::string getStatus ()` override
Returns a string representing the current state of the implementing component.

- `std::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

- `std::map< std::vector< dirCoordinate >, functionValue >` `getValues` (const `std::list< std::vector< dirCoordinate >>` &points)
Returns the function values at the given points.
- `std::list< std::shared_ptr< HyRect >>` `optimalRectangles` (size_t nrRects, functionValue phi)
Finds potentially optimal rectangles that should be divided in the current iteration.
- void `addActiveRects` (const `std::list< std::shared_ptr< HyRect >>` &rects)
Requests values at the corners of the given rectangles and add all given `HyRect` instances to `activeRects`.
- void `removeActiveRects` (const `std::list< std::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.

Static Private Member Functions

- static functionValue `estimatedValue` (const `std::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.
- `DirectStoppingCondition` `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.
- `std::map< depth, std::set< std::shared_ptr< HyRect >, CmpSharedHyrect >, std::greater<> >` `activeRects`
Holds all rectangles that are immediate part of the current partition.

Additional Inherited Members

8.13.1 Detailed Description

A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.
Definition at line 41 of file `DirectOptimizer.h`.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 DirectOptimizer()

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

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>	Controller to be used for evaluating the optimized function.
<i>params</i>	ParameterDefinition list to be optimized.
<i>con</i>	DirectStoppingCondition defining the end of optimization.
<i>trackProgress</i>	Defines whether the progress should be printed in a <code>.csv</code> file.

Definition at line 17 of file `DirectOptimizer.cpp`.
References [DirectOptimizer\(\)](#), [level](#), and [trackProgress](#).
Referenced by [DirectOptimizer\(\)](#).

8.13.3 Member Function Documentation

8.13.3.1 addActiveRects()

```
void DirectOptimizer::addActiveRects (
    const std::list< std::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 112 of file `DirectOptimizer.cpp`.

8.13.3.2 estimatedValue()

```
functionValue DirectOptimizer::estimatedValue (
    const std::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 92 of file `DirectOptimizer.cpp`.
References [HyRect::getAvgValue\(\)](#), and [HyRect::getDiagonalLength\(\)](#).

8.13.3.3 getName()

```
std::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 170 of file DirectOptimizer.cpp.

8.13.3.4 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 187 of file DirectOptimizer.cpp.

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

8.13.3.5 getStatus()

```
std::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 174 of file DirectOptimizer.cpp.

References [Levels::getLevel\(\)](#), [getPartitionSize\(\)](#), [iterations](#), and [level](#).

8.13.3.6 getStatusBar()

```
std::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 183 of file DirectOptimizer.cpp.

8.13.3.7 getValues()

```
std::map< std::vector< dirCoordinate >, functionValue > DirectOptimizer::getValues (
    const std::list< std::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 74 of file DirectOptimizer.cpp.

8.13.3.8 optimalRectangles()

```
std::list< std::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 activeRects).
<i>phi</i>	Value at the current minimum.

Returns

A list of potentially optimal rectangles.

Definition at line 96 of file DirectOptimizer.cpp.

8.13.3.9 removeActiveRects()

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

Removes the given rectangles from [activeRects](#).

Parameters

<i>rects</i>	Rectangles to be removed.
--------------	---------------------------

Definition at line 147 of file DirectOptimizer.cpp.

8.13.3.10 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 25 of file DirectOptimizer.cpp.

References [Optimizer::getValueMap\(\)](#), [Levels::isGlobal\(\)](#), [iterations](#), [Levels::L3_EPSILON](#), [level](#), [Levels::nextLevel\(\)](#), [saveProgress\(\)](#), [Levels::setGlobal\(\)](#), and [trackProgress](#).

8.13.3.11 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 activeRects).

Definition at line 158 of file `DirectOptimizer.cpp`.

Referenced by `runOptimization()`.

8.13.4 Member Data Documentation

8.13.4.1 activeRects

```
std::map<depth, std::set<std::shared_ptr<HyRect>, CmpSharedHyrect>, std::greater<> > Direct↔
Optimizer::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 74 of file `DirectOptimizer.h`.

8.13.4.2 D

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

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

Definition at line 46 of file `DirectOptimizer.h`.

8.13.4.3 iterations

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

Number of iterations completed.

Definition at line 50 of file `DirectOptimizer.h`.

Referenced by `getStatus()`, and `runOptimization()`.

8.13.4.4 level

```
Levels DirectOptimizer::level [private]
```

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

Definition at line 58 of file `DirectOptimizer.h`.

Referenced by `DirectOptimizer()`, `getStatus()`, and `runOptimization()`.

8.13.4.5 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 62 of file `DirectOptimizer.h`.

8.13.4.6 stopCon

`DirectStoppingCondition` `DirectOptimizer::stopCon` [private]

An object deciding when the optimization stops.

Definition at line 54 of file `DirectOptimizer.h`.

8.13.4.7 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 67 of file `DirectOptimizer.h`.

Referenced by `DirectOptimizer()`, and `runOptimization()`.

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

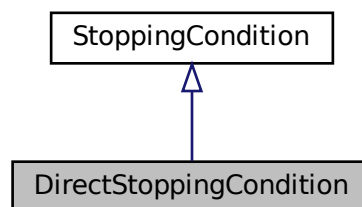
- `/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectOptimizer.h`
- `/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectOptimizer.cpp`

8.14 DirectStoppingCondition Class Reference

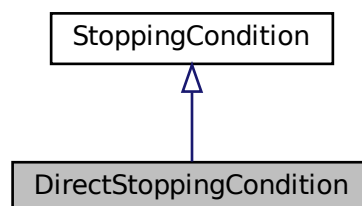
A class used for deciding whether the DIRECT should be stopped.

```
#include "DirectStoppingCondition.h"
```

Inheritance diagram for `DirectStoppingCondition`:



Collaboration diagram for `DirectStoppingCondition`:



Public Member Functions

- [DirectStoppingCondition](#) (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.
- [DirectStoppingCondition](#) (const nlohmann::json &stopCon)
Creates a [StoppingCondition](#) based on the given json configuration.
- bool [evaluate](#) (size_t evaluations, size_t hyrects, [functionValue](#) newBestVal)
Checks if any of the configured conditions is met for the given parameters.

Private Attributes

- const size_t [NR_HYRECTS](#)
Number of rectangles in the partition after which the optimization should stop.

Additional Inherited Members

8.14.1 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.

Author

Per Natzschka

Definition at line 22 of file [DirectStoppingCondition.h](#).

8.14.2 Constructor & Destructor Documentation

8.14.2.1 DirectStoppingCondition() [1/2]

```
DirectStoppingCondition::DirectStoppingCondition (
    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 ACCURACY).
<i>accuracyIterations</i>	Number of iterations used in accuracy condition (see NR_ACCURACY_ITERATIONS).

Definition at line 9 of file [DirectStoppingCondition.cpp](#).

References [StoppingCondition::StoppingCondition\(\)](#), and [NR_HYRECTS](#).

8.14.2.2 DirectStoppingCondition() [2/2]

```
DirectStoppingCondition::DirectStoppingCondition (
```

```
const nlohmann::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 16 of file DirectStoppingCondition.cpp.

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

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

8.14.3 Member Function Documentation

8.14.3.1 evaluate()

```
bool DirectStoppingCondition::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 22 of file DirectStoppingCondition.cpp.

References [StoppingCondition::evaluate\(\)](#), and [NR_HYRECTS](#).

8.14.4 Member Data Documentation

8.14.4.1 NR_HYRECTS

```
const size_t DirectStoppingCondition::NR_HYRECTS [private]
```

Number of rectangles in the partition after which the optimization should stop.

Definition at line 27 of file DirectStoppingCondition.h.

Referenced by [DirectStoppingCondition\(\)](#), and [evaluate\(\)](#).

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

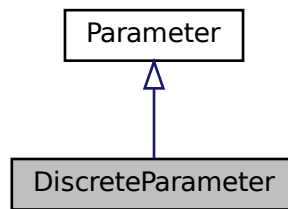
- </home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectStoppingCondition.h>
- </home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectStoppingCondition.cpp>

8.15 DiscreteParameter Class Reference

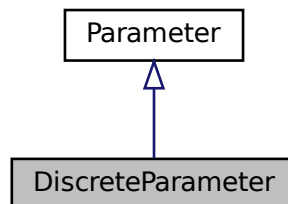
Implements a [Parameter](#) using discrete values.

```
#include "DiscreteParameter.h"
```

Inheritance diagram for DiscreteParameter:



Collaboration diagram for DiscreteParameter:



Public Member Functions

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

8.15.1 Detailed Description

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

The value of the [Parameter](#) is calculated as $val = times \cdot step + offset$.

Author

Per Natzschka

Definition at line 17 of file DiscreteParameter.h.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 DiscreteParameter() [1/2]

```
DiscreteParameter::DiscreteParameter (
    std::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>	ParameterDefinition of the Parameter .
<i>step</i>	Difference between discrete values.
<i>value</i>	Initial value of the Parameter .

Definition at line 13 of file DiscreteParameter.cpp.

References [Parameter::Parameter\(\)](#), [Parameter::getMax\(\)](#), [Parameter::getMin\(\)](#), [offset](#), [step](#), and [times](#).

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

8.15.2.2 DiscreteParameter() [2/2]

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

Creates a [DiscreteParameter](#) with the given [ParameterDefinition](#) and step.

Calculates [times](#) and [offset](#) automatically.

Parameters

<i>def</i>	ParameterDefinition of the Parameter .
<i>step</i>	Difference between discrete values.

Definition at line 23 of file DiscreteParameter.cpp.

References [DiscreteParameter\(\)](#), [Parameter::getMax\(\)](#), and [Parameter::getMin\(\)](#).

8.15.3 Member Function Documentation

8.15.3.1 `getOffset()`

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

Returns the value of [offset](#).

Returns

A floating point number representing the offset.

Definition at line 43 of file `DiscreteParameter.cpp`.

References [offset](#).

8.15.3.2 `getStep()`

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

Returns the value of [step](#).

Returns

A floating point number representing the difference between discrete values.

Definition at line 39 of file `DiscreteParameter.cpp`.

References [step](#).

8.15.3.3 `getTimes()`

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

Returns the value of [times](#).

Returns

An integer representing the times value.

Definition at line 27 of file `DiscreteParameter.cpp`.

References [times](#).

8.15.3.4 `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 47 of file `DiscreteParameter.cpp`.

References [offset](#), [step](#), and [times](#).

8.15.3.5 `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 31 of file DiscreteParameter.cpp.

References `Parameter::getMax()`, `Parameter::getMin()`, `offset`, `step`, and `times`.

Referenced by `setVal()`.

8.15.3.6 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 <code>DiscreteParameter</code> to.
------------	---

Implements `Parameter`.

Definition at line 51 of file DiscreteParameter.cpp.

References `offset`, `setTimes()`, and `step`.

8.15.4 Member Data Documentation**8.15.4.1 offset**

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

Offset used in the value calculation.

Definition at line 30 of file DiscreteParameter.h.

Referenced by `DiscreteParameter()`, `getOffset()`, `getVal()`, `setTimes()`, and `setVal()`.

8.15.4.2 step

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

Difference between discrete values.

Used in the value calculation.

Definition at line 26 of file DiscreteParameter.h.

Referenced by `DiscreteParameter()`, `getStep()`, `getVal()`, `setTimes()`, and `setVal()`.

8.15.4.3 times

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

Times used in the value calculation.

Definition at line 22 of file DiscreteParameter.h.

Referenced by `DiscreteParameter()`, `getTimes()`, `getVal()`, and `setTimes()`.

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

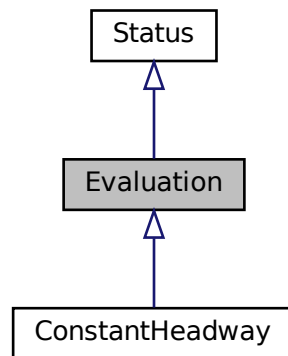
- [/home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParameter.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParameter.cpp](#)

8.16 Evaluation Class Reference

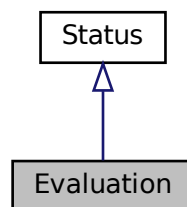
A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization.

```
#include "Evaluation.h"
```

Inheritance diagram for Evaluation:



Collaboration diagram for Evaluation:



Public Member Functions

- virtual `functionValue processOutput` (std::filesystem::path path, std::set< `runId` > experimentIds)=0
Returns a value to the results of a single simulation run.
- virtual std::map< std::pair< std::filesystem::path, std::set< `runId` > >, `functionValue` > `processOutput` (const std::set< std::pair< std::filesystem::path, std::set< `runId` >>> &experimentResults)
- std::string `getName` () override
Returns a string representing the name of the implementing component in natural language.
- std::string `getStatus` () override
Returns a string representing the current state of the implementing component.
- std::string `getStatusBar` () override
Returns a string representing the current progress of the calculations of the implementing component.

Additional Inherited Members

8.16.1 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.

Author

Per Natzschka

Definition at line 33 of file Evaluation.h.

8.16.2 Member Function Documentation

8.16.2.1 getName()

```
std::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 18 of file Evaluation.cpp.

References [Status::getName\(\)](#).

8.16.2.2 getStatus()

```
std::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 22 of file Evaluation.cpp.

References [Status::getStatus\(\)](#).

8.16.2.3 getStatusBar()

```
std::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 26 of file Evaluation.cpp.

References [Status::getStatusBar\(\)](#).

8.16.2.4 processOutput() [1/2]

```
std::map< std::pair< std::filesystem::path, std::set< runId > >, functionValue > Evaluation←
::processOutput (
    const std::set< std::pair< std::filesystem::path, std::set< runId >>> & experiment←
Results ) [virtual]
```

Returns values to the results of multiple simulation runs.

fi * Simply calls `processOutput(std::filesystem::path, std::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 10 of file `Evaluation.cpp`.

References `processOutput()`.

8.16.2.5 processOutput() [2/2]

```
virtual functionValue Evaluation::processOutput (
    std::filesystem::path path,
    std::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](#)

8.17 GrahamScan Class Reference

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

```
#include "GrahamScan.h"
```

Collaboration diagram for GrahamScan:

GrahamScan

Static Public Member Functions

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

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

8.17.1 Detailed Description

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

Author

Per Natzschka

Definition at line 19 of file GrahamScan.h.

8.17.2 Member Function Documentation

8.17.2.1 scan()

```
std::list< std::pair< std::shared_ptr< HyRect >, double > > GrahamScan::scan (
    std::list< std::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 13 of file GrahamScan.cpp.

References HyRect::getAvgValue(), HyRect::getDepth(), and HyRect::getDiagonalLength().

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

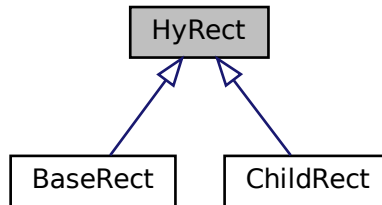
- /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/GrahamScan.h
- /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/GrahamScan.cpp

8.18 HyRect Class Reference

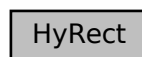
An abstract class representing a rectangular part of the search space.

```
#include "HyRect.h"
```

Inheritance diagram for HyRect:



Collaboration diagram for HyRect:



Public Member Functions

- [HyRect](#) (dimension *D*, position *pos*, depth *t*)
Creates a *HyRect* with the given dimensionality, position and depth.
- virtual `std::array< std::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 `std::array< std::shared_ptr< HyRect >, 3 > divide` (const `std::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](#).

8.18.1 Detailed Description

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

Author

Per Natzschka

Definition at line 44 of file [HyRect.h](#).

8.18.2 Constructor & Destructor Documentation

8.18.2.1 [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 12 of file [HyRect.cpp](#).

References [D](#), [pos](#), and [t](#).

Referenced by [BaseRect::BaseRect\(\)](#), and [ChildRect::ChildRect\(\)](#).

8.18.3 Member Function Documentation

8.18.3.1 divide()

```
std::array< std::shared_ptr< HyRect >, 3 > HyRect::divide (
    const std::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 15 of file HyRect.cpp.

References [ChildRect::ChildRect\(\)](#).

8.18.3.2 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 41 of file HyRect.cpp.

References [avgValue](#).

Referenced by [DirectOptimizer::estimatedValue\(\)](#), [CmpSharedHyrect::operator\(\)\(\)](#), and [GrahamScan::scan\(\)](#).

8.18.3.3 getD()

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

Returns the value of [D](#).

Returns

A dimension representing the number of dimensions of the rectangle.

Definition at line 73 of file HyRect.cpp.

References [D](#).

Referenced by [ChildRect::ChildRect\(\)](#).

8.18.3.4 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 37 of file HyRect.cpp.

References [t](#).

Referenced by [ChildRect::ChildRect\(\)](#), and [GrahamScan::scan\(\)](#).

8.18.3.5 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 22 of file [HyRect.cpp](#).

References [D](#), and [t](#).

Referenced by [DirectOptimizer::estimatedValue\(\)](#), and [GrahamScan::scan\(\)](#).

8.18.3.6 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 33 of file [HyRect.cpp](#).

References [pos](#).

Referenced by [ChildRect::operator==\(\)](#).

8.18.3.7 getSamplingVertices()

```
virtual std::array<std::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>\(\)\(\)](#).

8.18.3.8 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 29 of file [HyRect.cpp](#).

References [D](#), and [t](#).

Referenced by [ChildRect::getSamplingVertices\(\)](#).

8.18.3.9 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>	HyRect to be compared.
------------	--

Returns

A boolean defining if the [HyRect](#) objects have different positions in the partition tree.

Definition at line 57 of file [HyRect.cpp](#).

References [operator==\(\(\)\)](#).

8.18.3.10 operator<()

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

Compares depth [t](#) and [avgValue](#) of the given [HyRect](#) objects.

Parameters

<i>rect</i>	HyRect to be compared.
-------------	--

Returns

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

Definition at line 53 of file [HyRect.cpp](#).

References [avgValue](#), and [t](#).

Referenced by [operator<=\(\)](#), [operator>\(\)](#), and [operator>=\(\)](#).

8.18.3.11 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>	HyRect to be compared.
------------	--

Returns

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

Definition at line 65 of file [HyRect.cpp](#).

References [operator<\(\)](#).

8.18.3.12 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

<i>rect</i>	HyRect to be compared.
-------------	------------------------

Returns

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

Reimplemented in ChildRect.

Definition at line 49 of file HyRect.cpp.

References D, pos, and t.

Referenced by operator!=(*l*).

8.18.3.13 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

<i>rhs</i>	HyRect to be compared.
------------	------------------------

Returns

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

Definition at line 61 of file HyRect.cpp.

References operator<(*l*).

8.18.3.14 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>	HyRect to be compared.
------------	------------------------

Returns

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

Definition at line 69 of file HyRect.cpp.

References operator<(*l*).

8.18.3.15 setAvgValue()

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

Sets the value of avgValue.

Parameters

<i>value</i>	Average value sampled at the corners of the rectangle.
--------------	--

Definition at line 45 of file HyRect.cpp.

References avgValue.

8.18.4 Member Data Documentation

8.18.4.1 avgValue

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

Mean between the values obtained at the parameters returned by [getSamplingVertices](#).

Definition at line 64 of file HyRect.h.

Referenced by [getAvgValue\(\)](#), [operator<\(\)](#), and [setAvgValue\(\)](#).

8.18.4.2 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 50 of file HyRect.h.

Referenced by [HyRect\(\)](#), [getD\(\)](#), [getDiagonalLength\(\)](#), [BaseRect::getSamplingVertices\(\)](#), [getSplitDim\(\)](#), and [operator==\(\)](#).

8.18.4.3 pos

`position` HyRect::pos [protected]

Position of the rectangle relative to its parent rectangle.

For [BaseRect](#), *pos* is always BASE.

Definition at line 60 of file HyRect.h.

Referenced by [HyRect\(\)](#), [getPos\(\)](#), [ChildRect::getSamplingVertices\(\)](#), [operator==\(\)](#), and [ChildRect::operator==\(\)](#).

8.18.4.4 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 55 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](#)

8.19 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"
```

Collaboration diagram for Levels:



Public Member Functions

- unsigned char `nextLevel` ()
Switches `currentLevel` to the next local level if `global` is false.
- `std::list< std::shared_ptr< HyRect > >` `getRectSubset` (const `std::map< depth, std::set< std::shared_ptr< HyRect >, CmpSharedHyrect >`, `std::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.

8.19.1 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 32 of file Levels.h.

8.19.2 Member Function Documentation

8.19.2.1 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 59 of file Levels.cpp.

References [getLevel\(\)](#), [L0_EPSILON](#), [L1_EPSILON](#), [L2_EPSILON](#), and [L3_EPSILON](#).

8.19.2.2 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 72 of file Levels.cpp.

References [currentLevel](#), and [global](#).

Referenced by [getEpsilon\(\)](#), [getRectSubset\(\)](#), [DirectOptimizer::getStatus\(\)](#), and [nextLevel\(\)](#).

8.19.2.3 getRectSubset()

```
std::list< std::shared_ptr< HyRect > > Levels::getRectSubset (
    const std::map< depth, std::set< std::shared_ptr< HyRect >, CmpSharedHyrect >,
    std::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 HyRect of the current partition grouped by HyRect::t and sorted by HyRect::avgValue .
<i>size</i>	Number of HyRect in the given partition.

Returns

A list containing only the best [HyRect](#) per diagonal length in the subset based on the current level.

Definition at line 21 of file Levels.cpp.

References [getLevel\(\)](#), [global](#), [L0_SIZE](#), [L1_SIZE](#), [L2_SIZE](#), and [L3_SIZE](#).

8.19.2.4 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 88 of file Levels.cpp.

References [global](#).

Referenced by [DirectOptimizer::runOptimization\(\)](#).

8.19.2.5 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 13 of file Levels.cpp.

References [currentLevel](#), [getLevel\(\)](#), and [global](#).

Referenced by [DirectOptimizer::runOptimization\(\)](#).

8.19.2.6 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 92 of file Levels.cpp.

References [global](#).

Referenced by [DirectOptimizer::runOptimization\(\)](#).

8.19.3 Member Data Documentation

8.19.3.1 currentLevel

```
level Levels::currentLevel = 12_0 [private]
```

Local level the optimization is currently using when [global](#) is *false*.

Definition at line 37 of file Levels.h.

Referenced by [getLevel\(\)](#), and [nextLevel\(\)](#).

8.19.3.2 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 41 of file Levels.h.

Referenced by `getLevel()`, `getRectSubset()`, `isGlobal()`, `nextLevel()`, and `setGlobal()`.

8.19.3.3 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 59 of file Levels.h.

Referenced by `getEpsilon()`.

8.19.3.4 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 76 of file Levels.h.

Referenced by `getRectSubset()`.

8.19.3.5 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 55 of file Levels.h.

Referenced by `getEpsilon()`.

8.19.3.6 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 72 of file Levels.h.

Referenced by `getRectSubset()`.

8.19.3.7 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 51 of file Levels.h.

Referenced by `getEpsilon()`.

8.19.3.8 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 68 of file Levels.h.

Referenced by `getRectSubset()`.

8.19.3.9 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 47 of file Levels.h.

Referenced by `getEpsilon()`, and `DirectOptimizer::runOptimization()`.

8.19.3.10 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 64 of file Levels.h.

Referenced by `getRectSubset()`.

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

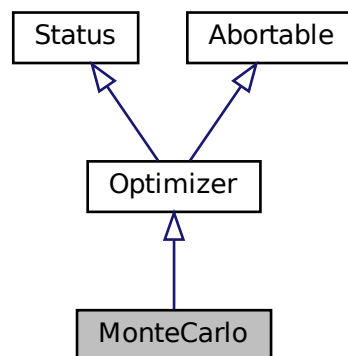
- `/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Levels.h`
- `/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Levels.cpp`

8.20 MonteCarlo Class Reference

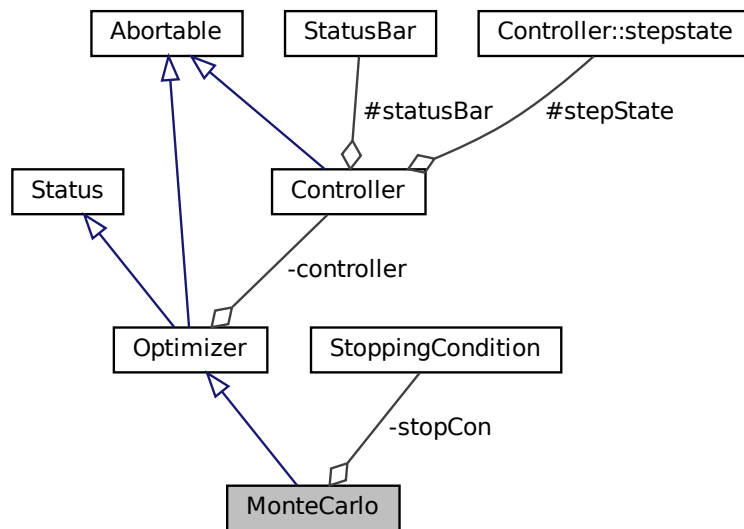
A class capable of finding the minimum of a blackbox function using the Monte Carlo algorithm.

```
#include "MonteCarlo.h"
```

Inheritance diagram for MonteCarlo:



Collaboration diagram for MonteCarlo:



Public Member Functions

- [MonteCarlo](#) ([Controller](#) &ctrl, const std::list< std::shared_ptr< [ParameterDefinition](#) >> ¶ms, nlohmann::json config)
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 Monte Carlo algorithm.
- std::string [getName](#) () override
Returns a string representing the name of the implementing component in natural language.
- std::string [getStatus](#) () override
Returns a string representing the current state of the implementing component.
- std::string [getStatusBar](#) () override
Returns a string representing the current progress of the calculations of the implementing component.

Private Member Functions

- void [saveProgress](#) ([functionValue](#) bestVal, size_t evaluations) const
Prints the current number of iterations, evaluations, rectangles and the current optimal value to a `.csv` file.

Private Attributes

- size_t [iterations](#) = 1
Number of iterations completed.
- const size_t [parallelTrials](#)
Number of (typically parallel executed) parameter combinations per iteration.
- [StoppingCondition](#) [stopCon](#)
An object deciding when the optimization stops.
- 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.

- `std::string` [lastEvaluations](#)

A string for logging purposes containing the evaluation of the simulation runs of the last iteration.

Additional Inherited Members

8.20.1 Detailed Description

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

Author

Burkhard Hensel

Definition at line 35 of file MonteCarlo.h.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 MonteCarlo()

```
MonteCarlo::MonteCarlo (
    Controller & ctrl,
    const std::list< std::shared_ptr< ParameterDefinition >> & params,
    nlohmann::json config )
```

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

Parameters

<code>ctrl</code>	Controller to be used for evaluating the optimized function.
<code>params</code>	ParameterDefinition list to be optimized.
<code>config</code>	Configuration parameters as JSON.

Definition at line 15 of file MonteCarlo.cpp.

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

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

8.20.3 Member Function Documentation

8.20.3.1 getName()

```
std::string MonteCarlo::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 83 of file MonteCarlo.cpp.

8.20.3.2 getStatus()

```
std::string MonteCarlo::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 87 of file MonteCarlo.cpp.

References iterations, and lastEvaluations.

8.20.3.3 getStatusBar()

```
std::string MonteCarlo::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 102 of file MonteCarlo.cpp.

8.20.3.4 runOptimization()

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

Starts the optimization using the Monte Carlo 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 22 of file MonteCarlo.cpp.

References [Optimizer::getValueMap\(\)](#), iterations, lastEvaluations, parallelTrials, saveProgress(), and trackProgress.

8.20.3.5 saveProgress()

```
void MonteCarlo::saveProgress (
    functionValue bestVal,
    size_t evaluations ) 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.

Definition at line 71 of file MonteCarlo.cpp.

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

8.20.4 Member Data Documentation

8.20.4.1 iterations

```
size_t MonteCarlo::iterations = 1 [private]
```

Number of iterations completed.
Definition at line 40 of file MonteCarlo.h.
Referenced by getStatus(), and runOptimization().

8.20.4.2 lastEvaluations

```
std::string MonteCarlo::lastEvaluations [private]
```

A string for logging purposes containing the evaluation of the simulation runs of the last iteration.
Definition at line 57 of file MonteCarlo.h.
Referenced by getStatus(), and runOptimization().

8.20.4.3 parallelTrials

```
const size_t MonteCarlo::parallelTrials [private]
```

Number of (typically parallel executed) parameter combinations per iteration.
Definition at line 44 of file MonteCarlo.h.
Referenced by runOptimization().

8.20.4.4 stopCon

```
StoppingCondition MonteCarlo::stopCon [private]
```

An object deciding when the optimization stops.
Definition at line 48 of file MonteCarlo.h.

8.20.4.5 trackProgress

```
bool MonteCarlo::trackProgress [private]
```

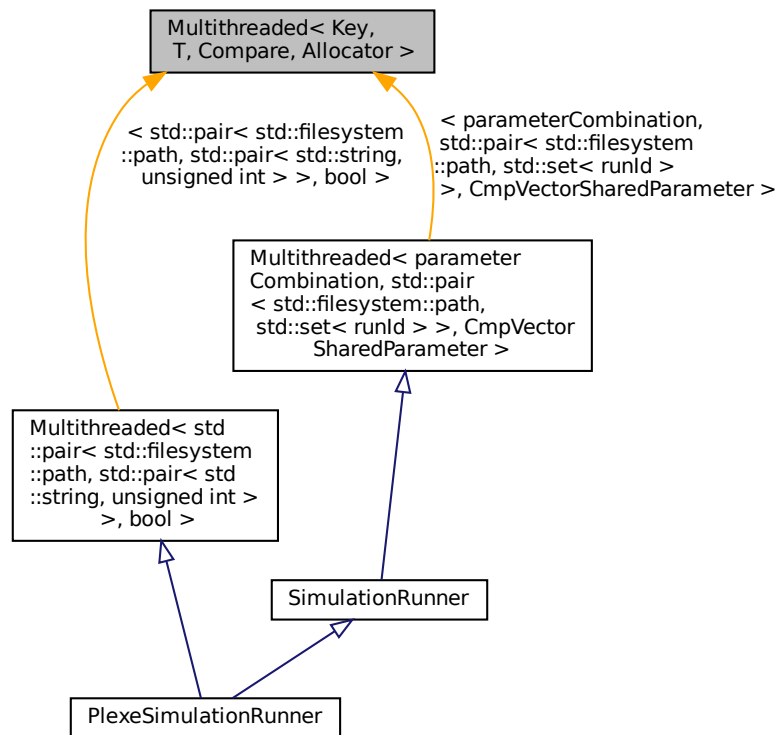
Defines if the current number of iterations, evaluations, rectangles and the optimal value should be recorded into a .csv file after each iteration.
Can be set in config.
Definition at line 53 of file MonteCarlo.h.
Referenced by runOptimization().
The documentation for this class was generated from the following files:

- [/home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/MonteCarlo.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/MonteCarlo.cpp](#)

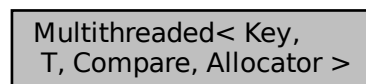
8.21 Multithreaded< Key, T, Compare, Allocator > Class Template Reference

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

Inheritance diagram for `Multithreaded< Key, T, Compare, Allocator >`:



Collaboration diagram for `Multithreaded< Key, T, Compare, Allocator >`:



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 `std::map< Key, T, Compare, Allocator > runMultithreadedFunctions` (`std::set< Key, Compare > runs`)
Pushes given tasks into `queue`, creates concurrent threads and merges them when execution is done.
- virtual `std::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.

8.21.1 Detailed Description

```
template<class Key, class T, class Compare = std::less<Key>, class Allocator = std::allocator<std::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

<i>Key</i>	Argument type of the concurrent <code>work</code> function.
<i>T</i>	Result type of the concurrent <code>work</code> function.
<i>Compare</i>	Comparison for objects of type Key.
<i>Allocator</i>	Allocator for pairs of constant Key and T.

Author

Per Natschka

Definition at line 27 of file Multithreaded.h.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 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 12 of file Multithreaded.hpp.

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

Referenced by `SimulationRunner::SimulationRunner()`.

8.21.3 Member Function Documentation

8.21.3.1 multithreadFunction()

```
template<class Key , class T , class Compare , class Allocator >
std::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 36 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()`.

8.21.3.2 runMultithreadedFunctions()

```
template<class Key , class T , class Compare , class Allocator >
std::map< Key, T, Compare, Allocator > Multithreaded< Key, T, Compare, Allocator >::run↵
MultithreadedFunctions (
    std::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 work should to be executed.
-------------	---

Returns

A map which maps arguments to their respective calculated values.

Definition at line 17 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()`.

8.21.3.3 work()

```
template<class Key , class T , class Compare = std::less<Key>, class Allocator = std::allocator<std↵
::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.

Implemented in [SimulationRunner](#), and [PlexeSimulationRunner](#).

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

8.21.4 Member Data Documentation**8.21.4.1 NR_THREADS**

```
template<class Key , class T , class Compare = std::less<Key>, class Allocator = std::allocator<std←
::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 40 of file `Multithreaded.h`.

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

8.21.4.2 queue

```
template<class Key , class T , class Compare = std::less<Key>, class Allocator = std::allocator<std←
::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 44 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:

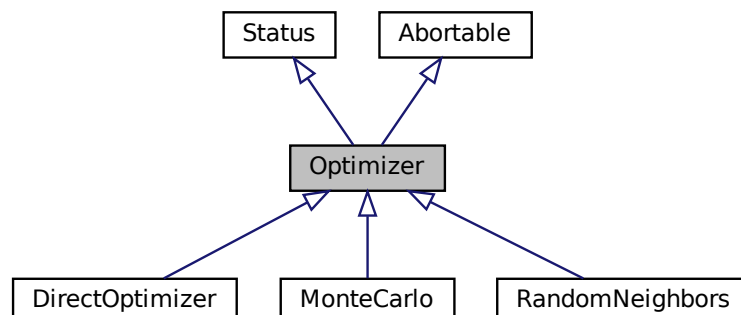
- `/home/runner/work/Simopticon/Simopticon/src/utils/Multithreaded.h`
- `/home/runner/work/Simopticon/Simopticon/src/utils/Multithreaded.tpp`

8.22 Optimizer Class Reference

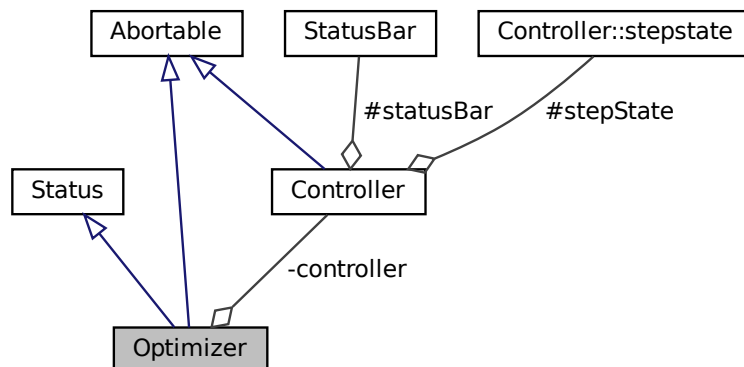
A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.

```
#include "Optimizer.h"
```

Inheritance diagram for Optimizer:



Collaboration diagram for `Optimizer`:



Public Member Functions

- `Optimizer` (`Controller` &ctrl, `std::list`< `std::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`.
- `std::string` `getName` () override
Returns a string representing the name of the implementing component in natural language.
- `std::string` `getStatus` () override
Returns a string representing the current state of the implementing component.
- `std::string` `getStatusBar` () override
Returns a string representing the current progress of the calculations of the implementing component.

Protected Member Functions

- `std::map`< `parameterCombination`, `functionValue` > `requestValues` (const `std::list`< `parameterCombination` > ¶ms)
Requests the values when using certain `parameterCombinations` from `controller`.

Protected Attributes

- `std::list`< `std::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

8.22.1 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 parameterCombinations 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.

Author

Per Natzschka

Definition at line 43 of file Optimizer.h.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 Optimizer()

```
Optimizer::Optimizer (
    Controller & ctrl,
    std::list< std::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>	Controller to be used for evaluation of parameterCombinations.
<i>params</i>	List of ParameterDefinition defining the parameters that must be optimized.

Definition at line 18 of file Optimizer.cpp.

References [controller](#), and [parameters](#).

8.22.3 Member Function Documentation

8.22.3.1 getName()

```
std::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 26 of file Optimizer.cpp.

References [Status::getName\(\)](#).

8.22.3.2 getStatus()

```
std::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 30 of file Optimizer.cpp.

References [Status::getStatus\(\)](#).

8.22.3.3 getStatusBar()

```
std::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 34 of file Optimizer.cpp.

References [Status::getStatusBar\(\)](#).

8.22.3.4 getValueMap()

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

Returns a reference to [Controller::valueMap](#).

Basically calls [Controller::getValueMap](#) on [controller](#).

Returns

Definition at line 22 of file Optimizer.cpp.

References [controller](#), and [Controller::getValueMap\(\)](#).

Referenced by [DirectOptimizer::runOptimization\(\)](#), [MonteCarlo::runOptimization\(\)](#), and [RandomNeighbors::runOptimization\(\)](#).

8.22.3.5 requestValues()

```
std::map< parameterCombination, functionValue > Optimizer::requestValues (
    const std::list< parameterCombination > & params ) [protected]
```

Requests the values when using certain parameterCombinations from [controller](#).

Basically calls [Controller::requestValues](#) with the given values.

Parameters

<i>params</i>	parameterCombinations to be evaluated.
---------------	--

Returns

A map which maps parameterCombinations to their respective values.

Definition at line 14 of file Optimizer.cpp.

References [controller](#), and [Controller::requestValues\(\)](#).

8.22.3.6 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 [RandomNeighbors](#), [MonteCarlo](#), and [DirectOptimizer](#).

Referenced by `Controller::run()`.

8.22.4 Member Data Documentation

8.22.4.1 controller

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

Reference to the executing [Controller](#) to be able to request values using [Controller::requestValues](#).

Definition at line 48 of file `Optimizer.h`.

Referenced by `Optimizer()`, `getValueMap()`, and `requestValues()`.

8.22.4.2 parameters

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

List of parameters to be optimized.

Definition at line 54 of file `Optimizer.h`.

Referenced by `Optimizer()`.

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

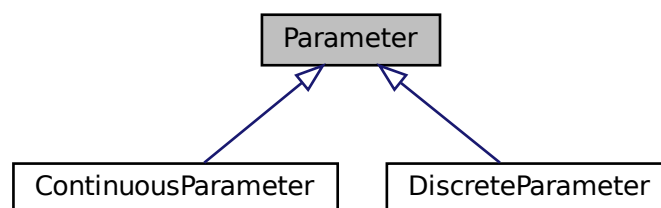
- `/home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.h`
- `/home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.cpp`

8.23 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`:



Collaboration diagram for Parameter:

Parameter

Public Member Functions

- [Parameter](#) (std::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 std::string & [getUnit](#) () const
 - Returns the unit string of the [Parameter](#) stored in [ParameterDefinition::unit](#) of [definition](#).*
- const std::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

- std::shared_ptr< [ParameterDefinition](#) > [definition](#)
 - Reference to the defining [ParameterDefinition](#).*

8.23.1 Detailed Description

A class acting as the container of the value of a parameter defined by a [ParameterDefinition](#).

Author

Per Natzschka

Definition at line 31 of file [Parameter.h](#).

8.23.2 Constructor & Destructor Documentation

8.23.2.1 Parameter()

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

Creates a [Parameter](#) with the given [ParameterDefinition](#).

Parameters

<i>def</i>	Definition of properties of the Parameter .
------------	---

Definition at line 13 of file Parameter.cpp.

References definition.

Referenced by [ContinuousParameter::ContinuousParameter\(\)](#), and [DiscreteParameter::DiscreteParameter\(\)](#).

8.23.3 Member Function Documentation

8.23.3.1 getConfig()

```
const std::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 29 of file Parameter.cpp.

References definition, and [ParameterDefinition::getConfig\(\)](#).

Referenced by [StatusBar::printResult\(\)](#).

8.23.3.2 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 21 of file Parameter.cpp.

References definition, and [ParameterDefinition::getMax\(\)](#).

Referenced by [ContinuousParameter::ContinuousParameter\(\)](#), [DiscreteParameter::DiscreteParameter\(\)](#), [Parameter↔Normalizer::normalize\(\)](#), [DiscreteParameter::setTimes\(\)](#), and [ContinuousParameter::setVal\(\)](#).

8.23.3.3 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 17 of file Parameter.cpp.

References definition, and [ParameterDefinition::getMin\(\)](#).

Referenced by [ContinuousParameter::ContinuousParameter\(\)](#), [DiscreteParameter::DiscreteParameter\(\)](#), [Parameter↔Normalizer::normalize\(\)](#), [DiscreteParameter::setTimes\(\)](#), and [ContinuousParameter::setVal\(\)](#).

8.23.3.4 getUnit()

```
const std::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 25 of file [Parameter.cpp](#).

References [definition](#), and [ParameterDefinition::getUnit\(\)](#).

Referenced by [StatusBar::printResult\(\)](#).

8.23.3.5 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\(\)](#).

8.23.3.6 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>	Parameter to be compared.
------------	---

Returns

A boolean defining if the [Parameter](#) objects contain another value or another [definition](#).

Definition at line 37 of file [Parameter.cpp](#).

References [operator==\(\(\)\)](#).

Referenced by [CmpVectorSharedParameter::operator\(\)\(\)](#).

8.23.3.7 operator<()

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

Compares the value of the given [Parameter](#) objects.

Parameters

<i>rhs</i>	Parameter to be compared.
------------	---

Returns

A boolean defining if the value of this [Parameter](#) is less than that of the given [Parameter](#).

Definition at line 41 of file `Parameter.cpp`.

References `getVal()`.

Referenced by `CmpVectorSharedParameter::operator()()`, `operator<=()`, `operator>()`, and `operator>=()`.

8.23.3.8 operator<=()

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

Compares the value of the given [Parameter](#) objects.

Basically negates `operator<`.

Parameters

<i>rhs</i>	Parameter to be compared.
------------	---

Returns

A boolean defining if the value of this [Parameter](#) is less than or equal to that of the given [Parameter](#).

Definition at line 49 of file `Parameter.cpp`.

References `operator<()`.

8.23.3.9 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>	Parameter to be compared.
------------	---

Returns

A boolean defining if the [Parameter](#) objects contain the same value for the same [definition](#).

Definition at line 33 of file `Parameter.cpp`.

References `definition`, and `getVal()`.

Referenced by `operator!=()`.

8.23.3.10 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>	Parameter to be compared.
------------	---

Returns

A boolean defining if the value of this [Parameter](#) is greater than that of the given [Parameter](#).

Definition at line 45 of file `Parameter.cpp`.

References `operator<()`.

8.23.3.11 operator>=()

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

Compares the value of the given [Parameter](#) objects.

Basically negates `operator>`.

Parameters

<i>rhs</i>	Parameter to be compared.
------------	---

Returns

A boolean defining if the value of this [Parameter](#) is greater than or equal to that of the given [Parameter](#).

Definition at line 53 of file `Parameter.cpp`.

References `operator<()`.

8.23.3.12 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 Parameter to.
------------	--

Implemented in [DiscreteParameter](#), and [ContinuousParameter](#).

8.23.4 Member Data Documentation**8.23.4.1 definition**

```
std::shared_ptr<ParameterDefinition> Parameter::definition [private]
```

Reference to the defining [ParameterDefinition](#).

Definition at line 36 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`

8.24 ParameterDefinition Class Reference

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

```
#include "ParameterDefinition.h"
```

Collaboration diagram for ParameterDefinition:

ParameterDefinition

Public Member Functions

- [ParameterDefinition](#) ([coordinate min](#), [coordinate max](#), `std::string config=""`, `std::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 std::string &getUnit` () const
Returns the unit string of the [Parameter](#) stored in [unit](#).
- `const std::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 std::string unit`
Unit of the [Parameter](#) (optional).
- `const std::string config`
String containing configuration details of the [Parameter](#) (optional).

8.24.1 Detailed Description

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

Author

Per Natzschka

Definition at line 18 of file `ParameterDefinition.h`.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 ParameterDefinition()

```
ParameterDefinition::ParameterDefinition (
    coordinate min,
    coordinate max,
    std::string config = "",
    std::string unit = "" )
```

Creates a [ParameterDefinition](#) with the given minimum, maximum, configuration string and unit.

Parameters

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

Definition at line 12 of file ParameterDefinition.cpp.
References config, max, min, and unit.

8.24.3 Member Function Documentation

8.24.3.1 getConfig()

```
const std::string & ParameterDefinition::getConfig ( ) const
```

Returns the configuration string of the [Parameter](#) stored in [config](#).

Returns

A string reference containing the configuration.

Definition at line 36 of file ParameterDefinition.cpp.
References config.
Referenced by [Parameter::getConfig\(\)](#).

8.24.3.2 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 28 of file ParameterDefinition.cpp.
References max.
Referenced by [ParameterNormalizer::denormalize\(\)](#), and [Parameter::getMax\(\)](#).

8.24.3.3 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 24 of file ParameterDefinition.cpp.
References min.
Referenced by [ParameterNormalizer::denormalize\(\)](#), and [Parameter::getMin\(\)](#).

8.24.3.4 getUnit()

```
const std::string & ParameterDefinition::getUnit ( ) const
```

Returns the unit string of the [Parameter](#) stored in [unit](#).

Returns

A string reference containing the unit.

Definition at line 32 of file ParameterDefinition.cpp.

References unit.

Referenced by Parameter::getUnit().

8.24.4 Member Data Documentation

8.24.4.1 config

```
const std::string ParameterDefinition::config [private]
```

String containing configuration details of the [Parameter](#) (optional).

May be used to transfer configuration information for [SimulationRunner](#).

Definition at line 36 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getConfig().

8.24.4.2 max

```
const coordinate ParameterDefinition::max [private]
```

Maximum value of the [Parameter](#).

Definition at line 27 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getMax().

8.24.4.3 min

```
const coordinate ParameterDefinition::min [private]
```

Minimum value of the [Parameter](#).

Definition at line 23 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getMin().

8.24.4.4 unit

```
const std::string ParameterDefinition::unit [private]
```

Unit of the [Parameter](#) (optional).

Definition at line 31 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](#)

8.25 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"
```

Collaboration diagram for ParameterNormalizer:

ParameterNormalizer

Public Member Functions

- [ParameterNormalizer](#) (std::list< std::shared_ptr< [ParameterDefinition](#) >> [parameters](#))
Creates a [ParameterNormalizer](#) with the given optimized parameters.
- [parameterCombination denormalize](#) (std::vector< [dirCoordinate](#) > cords)
Transforms the given point in the unit hypercube into a [parameterCombination](#).

Static Public Member Functions

- static std::vector< [dirCoordinate](#) > [normalize](#) (const [parameterCombination](#) ¶ms)
Transforms the given [parameterCombination](#) into a point in the unit hypercube.

Private Attributes

- std::list< std::shared_ptr< [ParameterDefinition](#) > > [parameters](#)
[ParameterDefinition](#) of the optimized parameters.

8.25.1 Detailed Description

A class used for transforming parameters between the actual [Parameter](#) space and the unit hypercube used in DIRECT algorithm.

Definition at line 25 of file [ParameterNormalizer.h](#).

8.25.2 Constructor & Destructor Documentation

8.25.2.1 ParameterNormalizer()

```
ParameterNormalizer::ParameterNormalizer (
    std::list< std::shared_ptr< ParameterDefinition >> parameters ) [explicit]
```

Creates a [ParameterNormalizer](#) with the given optimized parameters.

Parameters

<i>parameters</i>	ParameterDefinition of the optimized parameters.
-------------------	--

Definition at line 14 of file [ParameterNormalizer.cpp](#).

References [parameters](#).

8.25.3 Member Function Documentation

8.25.3.1 denormalize()

```
parameterCombination ParameterNormalizer::denormalize (
    std::vector< dirCoordinate > cords )
```

Transforms the given point in the unit hypercube into a parameterCombination.

Parameters

<i>cords</i>	Point in the unit hypercube to be transformed.
--------------	--

Returns

A parameterCombination corresponding to the given point in the unit hypercube.

Definition at line 26 of file ParameterNormalizer.cpp.

References ContinuousParameter::ContinuousParameter(), ParameterDefinition::getMax(), ParameterDefinition::getMin(), and parameters.

8.25.3.2 normalize()

```
std::vector< dirCoordinate > ParameterNormalizer::normalize (
    const parameterCombination & params ) [static]
```

Transforms the given parameterCombination into a point in the unit hypercube.

Parameters

<i>params</i>	parameterCombination to be transformed.
---------------	---

Returns

A point in the unit hypercube corresponding to the given parameterCombination.

Definition at line 18 of file ParameterNormalizer.cpp.

References Parameter::getMax(), Parameter::getMin(), and Parameter::getVal().

8.25.4 Member Data Documentation

8.25.4.1 parameters

```
std::list<std::shared_ptr<ParameterDefinition> > ParameterNormalizer::parameters [private]
```

[ParameterDefinition](#) of the optimized parameters.

Definition at line 30 of file ParameterNormalizer.h.

Referenced by ParameterNormalizer(), and denormalize().

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

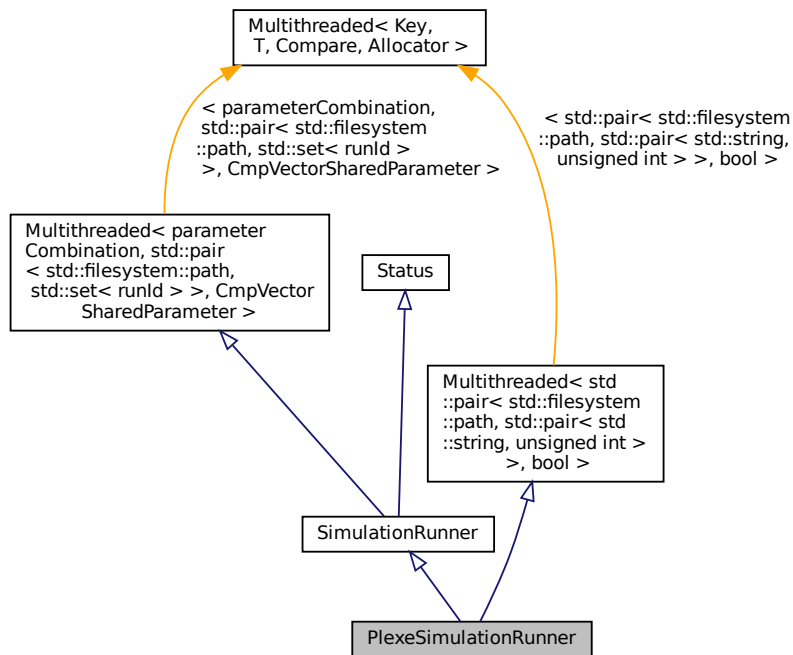
- /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/[ParameterNormalizer.h](#)
- /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/[ParameterNormalizer.cpp](#)

8.26 PlexeSimulationRunner Class Reference

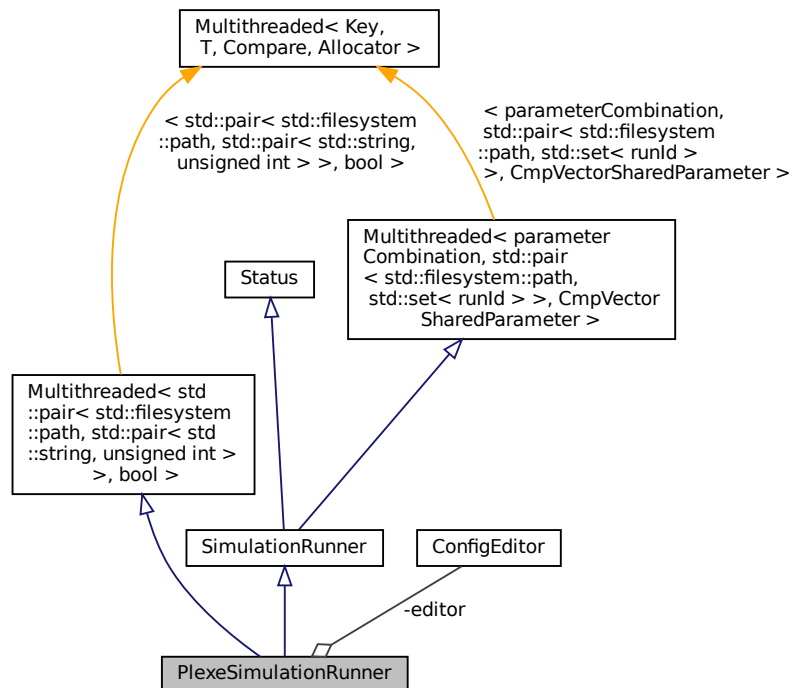
A class capable of starting platooning simulations in the [Plexe](#) framework with given parameterCombinations.

```
#include "PlexeSimulationRunner.h"
```

Inheritance diagram for PlexeSimulationRunner:



Collaboration diagram for PlexeSimulationRunner:



Public Member Functions

- [PlexeSimulationRunner](#) (unsigned int threads, unsigned int repeat, std::vector< std::string > scenarios, [ConfigEditor](#) editor)
Creates [PlexeSimulationRunner](#) which cannot use more than the given number of threads.
- std::string [getName](#) () override
Returns a string representing the name of the implementing component in natural language.
- std::string [getStatus](#) () override
Returns a string representing the current state of the implementing component.
- std::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 parameterCombination.
- std::pair< std::filesystem::path, std::set< runId > > [work](#) ([parameterCombination](#) run) override
Runs simulations for the given parameterCombination.
- bool [work](#) (std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned int >> arg) override
Executes one run of a parameterCombination (meaning repetition k of scenario c).

Private Attributes

- const unsigned int [REPEAT](#)
Number of repetitions per parameterCombination and scenario in [SCENARIOS](#).
- const std::vector< std::string > [SCENARIOS](#)
Scenarios that are simulated per parameterCombination.
- [ConfigEditor](#) editor
[ConfigEditor](#) used for automatically creating .ini files with given [Parameter](#) settings.
- size_t [runNumber](#) = 0
Identifier for each simulated parameterCombination.
- std::mutex [runNumberLock](#)
Threadlock to prevent race conditions on concurrent access of [runNumber](#).

Additional Inherited Members

8.26.1 Detailed Description

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

Author

Per Natzschka

Definition at line 31 of file PlexeSimulationRunner.h.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 PlexeSimulationRunner()

```
PlexeSimulationRunner::PlexeSimulationRunner (
    unsigned int threads,
    unsigned int repeat,
    std::vector< std::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 $t = \min(\text{threads}, \text{repeat} \cdot \text{size}(\text{scenarios}))$ concurrent threads for parallelization of `work(std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned int >>)`. For the parallelization of `work(parameterCombination)` $t' = \lfloor \text{threads} \div t \rfloor$ concurrent threads are used.

Parameters

<i>threads</i>	Maximum number of threads to be used.
<i>repeat</i>	Number of repetitions per parameterCombination and scenario.
<i>scenarios</i>	Scenarios to be simulated per parameterCombination.
<i>editor</i>	ConfigEditor to be used.

Definition at line 15 of file `PlexeSimulationRunner.cpp`.

References [PlexeSimulationRunner\(\)](#), and `REPEAT`.

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

8.26.3 Member Function Documentation

8.26.3.1 getName()

```
std::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 83 of file `PlexeSimulationRunner.cpp`.

8.26.3.2 getRunId()

```
size_t PlexeSimulationRunner::getRunId ( ) [private]
```

Returns an unique number which can be used to identify the results of a certain parameterCombination.

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 59 of file `PlexeSimulationRunner.cpp`.

References [runNumber](#), and [runNumberLock](#).

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

8.26.3.3 getStatus()

```
std::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 87 of file PlexeSimulationRunner.cpp.

References REPEAT, and runNumber.

8.26.3.4 getStatusBar()

```
std::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 100 of file PlexeSimulationRunner.cpp.

8.26.3.5 work() [1/2]

```
std::pair< std::filesystem::path, std::set< runId > > PlexeSimulationRunner::work (
    parameterCombination run ) [override], [private], [virtual]
```

Runs simulations for the given parameterCombination.

Creates a new `.ini` file for the parameterCombination. 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>	parameterCombination 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 24 of file PlexeSimulationRunner.cpp.

References getRunId().

8.26.3.6 work() [2/2]

```
bool PlexeSimulationRunner::work (
    std::pair< std::filesystem::path, std::pair< std::basic_string< char >, unsigned
int >> arg ) [override], [private]
```

Executes one run of a parameterCombination (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 68 of file PlexeSimulationRunner.cpp.

8.26.4 Member Data Documentation**8.26.4.1 editor**

```
ConfigEditor PlexeSimulationRunner::editor [private]
```

[ConfigEditor](#) used for automatically creating .ini files with given [Parameter](#) settings.

Definition at line 48 of file PlexeSimulationRunner.h.

8.26.4.2 REPEAT

```
const unsigned int PlexeSimulationRunner::REPEAT [private]
```

Number of repetitions per parameterCombination and scenario in [SCENARIOS](#).

Translates to repeat setting in omnetpp.ini. Can be set in configuration.

Definition at line 38 of file PlexeSimulationRunner.h.

Referenced by PlexeSimulationRunner(), and getStatus().

8.26.4.3 runNumber

```
size_t PlexeSimulationRunner::runNumber = 0 [private]
```

Identifier for each simulated parameterCombination.

Is incremented when new parameterCombination is simulated. Used for unique directory names for result files.

Definition at line 54 of file PlexeSimulationRunner.h.

Referenced by getRunId(), and getStatus().

8.26.4.4 runNumberLock

```
std::mutex PlexeSimulationRunner::runNumberLock [private]
```

Threadlock to prevent race conditions on concurrent access of [runNumber](#).

Definition at line 58 of file PlexeSimulationRunner.h.

Referenced by getRunId().

8.26.4.5 SCENARIOS

```
const std::vector<std::string> PlexeSimulationRunner::SCENARIOS [private]
```

Scenarios that are simulated per parameterCombination.

Should not invoke a GUI (e.g. pick BrakingNoGui instead of Braking). Can be set in configuration.

Definition at line 43 of file PlexeSimulationRunner.h.

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

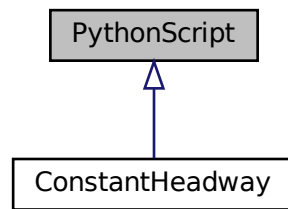
- /home/runner/work/Simopticon/Simopticon/src/runner/plexer/[PlexeSimulationRunner.h](#)
- /home/runner/work/Simopticon/Simopticon/src/runner/plexer/[PlexeSimulationRunner.cpp](#)

8.27 PythonScript Class Reference

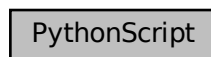
A class containing functionality for interfacing with the function of a Python module on creation.

```
#include "PythonScript.h"
```

Inheritance diagram for PythonScript:



Collaboration 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.

8.27.1 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.

Author

Per Natzschka

Definition at line 24 of file PythonScript.h.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 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 12 of file PythonScript.cpp.

Referenced by ConstantHeadway::ConstantHeadway().

8.27.2.2 ~PythonScript()

```
PythonScript::~PythonScript ( )
```

Ends connection to function [pFunc](#) and module [pModule](#).

Definition at line 36 of file PythonScript.cpp.

8.27.3 Member Data Documentation

8.27.3.1 pFunc

```
PyObject* PythonScript::pFunc [protected]
```

Pointer to function which should be used by the class.

Definition at line 33 of file PythonScript.h.

8.27.3.2 pModule

```
PyObject* PythonScript::pModule [protected]
```

Pointer to module that contains function which should be used by the class.

Definition at line 29 of file PythonScript.h.

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

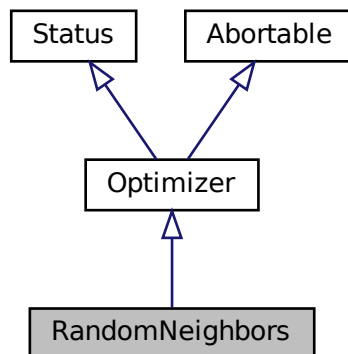
- [/home/runner/work/Simopticon/Simopticon/src/Utils/PythonScript.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/Utils/PythonScript.cpp](#)

8.28 RandomNeighbors Class Reference

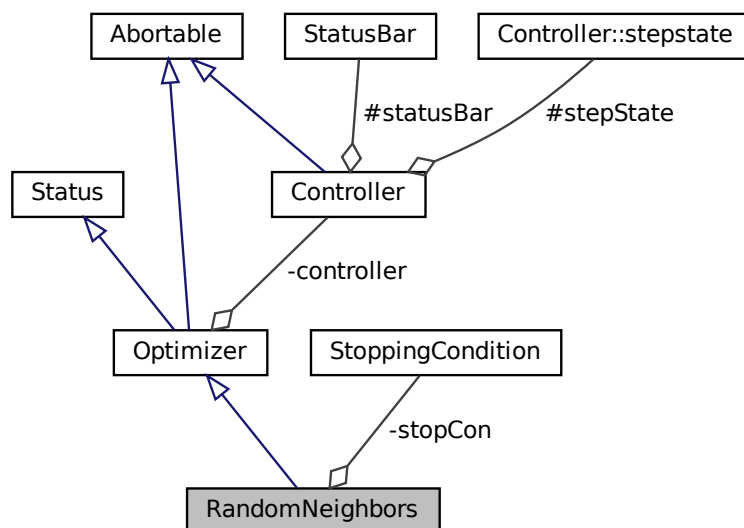
A class capable of finding the minimum of a blackbox function using the Random Neighbors algorithm.

```
#include "RandomNeighbors.h"
```

Inheritance diagram for RandomNeighbors:



Collaboration diagram for RandomNeighbors:



Public Member Functions

- `RandomNeighbors` (`Controller` &ctrl, const std::list< std::shared_ptr< `ParameterDefinition` >> ¶ms, nlohmann::json config)
Creates a `RandomNeighbors` optimizer 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.
- std::string `getName` () override
Returns a string representing the name of the implementing component in natural language.

- `std::string getStatus ()` override
Returns a string representing the current state of the implementing component.
- `std::string getStatusBar ()` override
Returns a string representing the current progress of the calculations of the implementing component.

Private Member Functions

- `void saveProgress (functionValue bestVal, size_t evaluations) const`
Prints the current number of iterations, evaluations, rectangles and the current optimal value to a .csv file.

Private Attributes

- `size_t iterations = 1`
Number of iterations completed.
- `const size_t parallelTrials`
Number of (typically parallel executed) parameter combinations per iteration.
- `const double neighborhoodWidth`
The search width around the current optimum relative to each parameter bounds pair.
- `const double localSearchProbability`
The probability for searching in the neighborhood instead of globally.
- `bool lastLocal = false`
True, if the last iteration was a local search, false if global search.
- `StoppingCondition stopCon`
An object deciding when the optimization stops.
- `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.
- `std::string lastEvaluations`
A string for logging purposes containing the evaluation of the simulation runs of the last iteration.

Additional Inherited Members

8.28.1 Detailed Description

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

Author

Burkhard Hensel

Definition at line 37 of file RandomNeighbors.h.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 RandomNeighbors()

```
RandomNeighbors::RandomNeighbors (
    Controller & ctrl,
    const std::list< std::shared_ptr< ParameterDefinition >> & params,
    nlohmann::json config )
```

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

Parameters

<i>ctrl</i>	Controller to be used for evaluating the optimized function.
<i>params</i>	ParameterDefinition list to be optimized.
<i>config</i>	Configuration parameters as JSON.

Definition at line 15 of file RandomNeighbors.cpp.

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

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

8.28.3 Member Function Documentation

8.28.3.1 getName()

```
std::string RandomNeighbors::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 105 of file RandomNeighbors.cpp.

8.28.3.2 getStatus()

```
std::string RandomNeighbors::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 109 of file RandomNeighbors.cpp.

References [iterations](#), [lastEvaluations](#), and [lastLocal](#).

8.28.3.3 getStatusBar()

```
std::string RandomNeighbors::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 126 of file RandomNeighbors.cpp.

8.28.3.4 runOptimization()

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

Starts the optimization.

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 26 of file `RandomNeighbors.cpp`.

References `Optimizer::getValueMap()`, `iterations`, `lastEvaluations`, `lastLocal`, `localSearchProbability`, `parallelTrials`, `saveProgress()`, and `trackProgress`.

8.28.3.5 saveProgress()

```
void RandomNeighbors::saveProgress (
    functionValue bestVal,
    size_t evaluations ) 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.

Definition at line 93 of file `RandomNeighbors.cpp`.

Referenced by `runOptimization()`.

8.28.4 Member Data Documentation

8.28.4.1 iterations

```
size_t RandomNeighbors::iterations = 1 [private]
```

Number of iterations completed.

Definition at line 42 of file `RandomNeighbors.h`.

Referenced by `getStatus()`, and `runOptimization()`.

8.28.4.2 lastEvaluations

```
std::string RandomNeighbors::lastEvaluations [private]
```

A string for logging purposes containing the evaluation of the simulation runs of the last iteration.

Definition at line 71 of file `RandomNeighbors.h`.

Referenced by `getStatus()`, and `runOptimization()`.

8.28.4.3 lastLocal

```
bool RandomNeighbors::lastLocal = false [private]
```

True, if the last iteration was a local search, false if global search.

Definition at line 58 of file `RandomNeighbors.h`.

Referenced by `getStatus()`, and `runOptimization()`.

8.28.4.4 localSearchProbability

```
const double RandomNeighbors::localSearchProbability [private]
```

The probability for searching in the neighborhood instead of globally.

Definition at line 54 of file RandomNeighbors.h.
Referenced by runOptimization().

8.28.4.5 neighborhoodWidth

```
const double RandomNeighbors::neighborhoodWidth [private]
```

The search width around the current optimum relative to each parameter bounds pair.
Definition at line 50 of file RandomNeighbors.h.

8.28.4.6 parallelTrials

```
const size_t RandomNeighbors::parallelTrials [private]
```

Number of (typically parallel executed) parameter combinations per iteration.
Definition at line 46 of file RandomNeighbors.h.
Referenced by runOptimization().

8.28.4.7 stopCon

```
StoppingCondition RandomNeighbors::stopCon [private]
```

An object deciding when the optimization stops.
Definition at line 62 of file RandomNeighbors.h.

8.28.4.8 trackProgress

```
bool RandomNeighbors::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 67 of file RandomNeighbors.h.

Referenced by runOptimization().

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

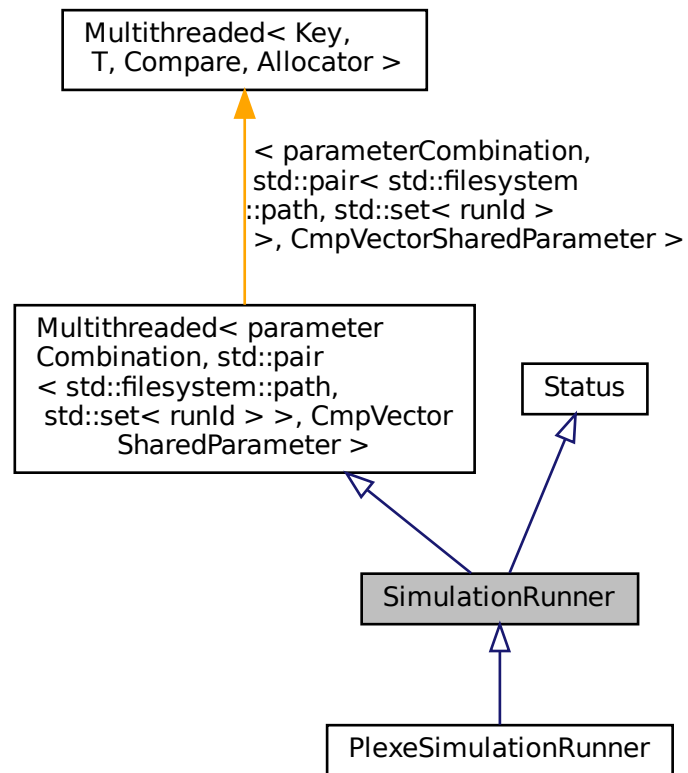
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors/RandomNeighbors.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors/RandomNeighbors.cpp](#)

8.29 SimulationRunner Class Reference

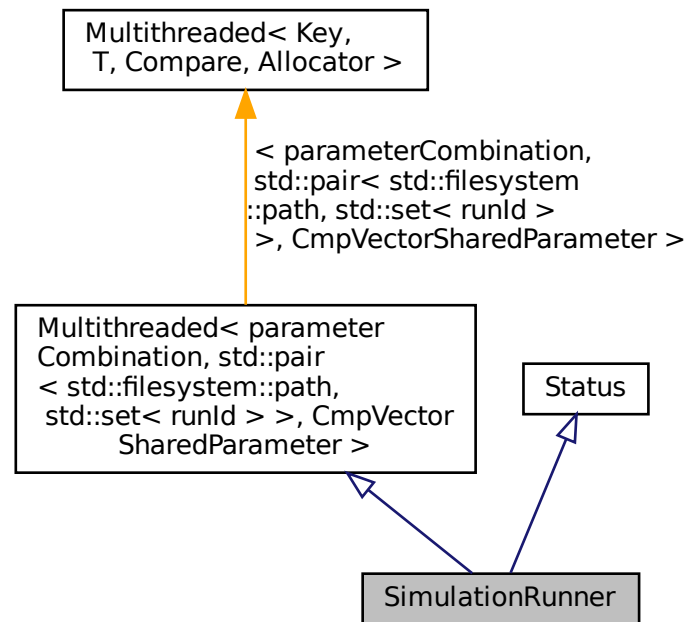
A class capable of running simulations with certain parameterCombinations.

```
#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 parameterCombinations concurrently.*
- virtual std::map< [parameterCombination](#), std::pair< std::filesystem::path, std::set< [runId](#) > >, [CmpVectorSharedParameter](#) > [runSimulations](#) (const std::set< [parameterCombination](#), [CmpVectorSharedParameter](#) > &runs)
 - Simulates the given parameterCombinations concurrently and returns their respective results.*
- std::string [getName](#) () override
 - Returns a string representing the name of the implementing component in natural language.*
- std::string [getStatus](#) () override
 - Returns a string representing the current state of the implementing component.*
- std::string [getStatusBar](#) () override
 - Returns a string representing the current progress of the calculations of the implementing component.*

Private Member Functions

- std::pair< std::filesystem::path, std::set< [runId](#) > > [work](#) ([parameterCombination](#) run) override=0
 - Deals with the simulation of a single parameterCombination.*

Additional Inherited Members

8.29.1 Detailed Description

A class capable of running simulations with certain parameterCombinations.

Author

Per Natzschka

Definition at line 39 of file SimulationRunner.h.

8.29.2 Constructor & Destructor Documentation**8.29.2.1 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 12 of file SimulationRunner.cpp.

References [Multithreaded< Key, T, Compare, Allocator >::Multithreaded\(\)](#).

8.29.3 Member Function Documentation**8.29.3.1 getName()**

```
std::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 20 of file SimulationRunner.cpp.

References [Status::getName\(\)](#).

8.29.3.2 getStatus()

```
std::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 24 of file SimulationRunner.cpp.

References [Status::getStatus\(\)](#).

8.29.3.3 getStatusBar()

```
std::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 28 of file SimulationRunner.cpp.

References [Status::getStatusBar\(\)](#).

8.29.3.4 runSimulations()

```
std::map< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, CmpVectorSharedParameter
> SimulationRunner::runSimulations (
```

```
    const std::set< parameterCombination, CmpVectorSharedParameter > & runs ) [virtual]
```

Simulates the given parameterCombinations 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 parameterCombinations to be simulated.
-------------	---

Returns

A map which maps the given parameterCombinations to their respective result directory and runIds.

Definition at line 16 of file SimulationRunner.cpp.

References [Multithreaded< Key, T, Compare, Allocator >::runMultithreadedFunctions\(\)](#).

8.29.3.5 work()

```
std::pair<std::filesystem::path, std::set<runId> > SimulationRunner::work (
    parameterCombination run ) [override], [private], [pure virtual]
```

Deals with the simulation of a single parameterCombination.

Overrides [Multithreaded::work](#) and therefore can be executed concurrently.

Parameters

<i>run</i>	parameterCombination to be simulated.
------------	---------------------------------------

Returns

A pair containing a path to the result directory and a set of runIds identifying the respective simulation runs.

Implements [Multithreaded< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, CmpVectorSharedParameter](#)

Implemented in [PlexeSimulationRunner](#).

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

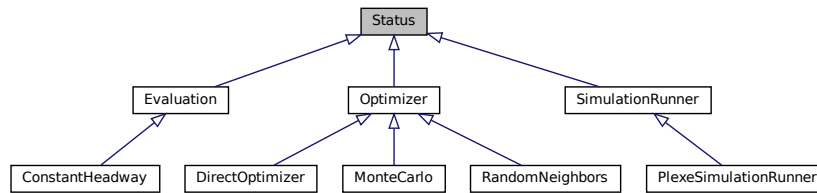
- [/home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.cpp](#)

8.30 Status Class Reference

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

```
#include "Status.h"
```

Inheritance diagram for Status:



Collaboration diagram for Status:



Public Member Functions

- virtual std::string [getName](#) ()
Returns a string representing the name of the implementing component in natural language.
- virtual std::string [getStatus](#) ()
Returns a string representing the current state of the implementing component.
- virtual std::string [getStatusBar](#) ()
Returns a string representing the current progress of the calculations of the implementing component.

Static Protected Attributes

- static const std::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 std::string [NO_NAME](#) = "No name specified"
Default message returned by [getName](#) if the implementing class does not override the function.

8.30.1 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.

Author

Per Natzschka

Definition at line 27 of file Status.h.

8.30.2 Member Function Documentation

8.30.2.1 getName()

```
std::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](#), [RandomNeighbors](#), [Optimizer](#), [MonteCarlo](#), [DirectOptimizer](#), [Evaluation](#), and [ConstantHeadway](#).

Definition at line 22 of file Status.cpp.

References `NO_NAME`.

Referenced by `Evaluation::getName()`, `Optimizer::getName()`, `SimulationRunner::getName()`, and `StatusBar::print←Status()`.

8.30.2.2 getStatus()

```
std::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](#), [RandomNeighbors](#), [Optimizer](#), [MonteCarlo](#), [DirectOptimizer](#), [Evaluation](#), and [ConstantHeadway](#).

Definition at line 14 of file Status.cpp.

References `NO_STATUS_SUPPORT`.

Referenced by `Evaluation::getStatus()`, `Optimizer::getStatus()`, `SimulationRunner::getStatus()`, and `StatusBar←::printStatus()`.

8.30.2.3 getStatusBar()

```
std::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](#), [RandomNeighbors](#), [Optimizer](#), [MonteCarlo](#), [DirectOptimizer](#), [Evaluation](#), and [ConstantHeadway](#).

Definition at line 18 of file Status.cpp.

References `NO_STATUS_SUPPORT`.

Referenced by `Evaluation::getStatusBar()`, `Optimizer::getStatusBar()`, `SimulationRunner::getStatusBar()`, and `StatusBar::updateStatus()`.

8.30.3 Member Data Documentation

8.30.3.1 NO_NAME

```
const std::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 36 of file Status.h.

Referenced by `getName()`.

8.30.3.2 NO_STATUS_SUPPORT

```
const std::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 32 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](#)

8.31 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"
```

Collaboration diagram for [StatusBar](#):



Public Member Functions

- void [updateStatus](#) ([Status](#) *opt, [Status](#) *runner, [Status](#) *eval, const std::pair< [parameterCombination](#), [functionValue](#) > ¤tVal, 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](#) (std::list< std::pair< [parameterCombination](#), std::pair< [functionValue](#), std::pair< [parameterCombination](#), [functionValue](#) > >>> top)

Prints the given parameterCombinations and respective values to command line.

Static Private Member Functions

- static void [printResult](#) (const [parameterCombination](#) &cords, [functionValue](#) optimum)
- static void [printStatus](#) ([Status](#) *object)

Prints the given result command line.

Prints the [Status](#) of the given object to the command line using [Status::getStatus](#).

Private Attributes

- std::pair< [parameterCombination](#), [functionValue](#) > lastVal

Pair of parameterCombination 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.

- `std::string lastStatus`

Last values of the `StatusBar` output (excluding value returned by `Status::getStatusBar`)

Static Private Attributes

- static const `std::string LARGE_DIVIDER` = `"\n\n" + std::string(70, '#') + "\n"`
Large divider used to visibly divide two sections of content.
- static const `std::string SMALL_DIVIDER` = `std::string(70, '-') + "\n"`
Small divider used to visibly divide two sections of content.

8.31.1 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 38 of file `StatusBar.h`.

8.31.2 Member Function Documentation

8.31.2.1 printResult()

```
void StatusBar::printResult (
    const parameterCombination & cords,
    functionValue optimum ) [static], [private]
```

Prints the given result command line.

Parameters

<code>cords</code>	parameterCombination of the given result.
<code>optimum</code>	Value of the given result.

Definition at line 56 of file `StatusBar.cpp`.

References `Parameter::getConfig()`, `Parameter::getUnit()`, and `Parameter::getVal()`.

Referenced by `printResults()`, and `updateStatus()`.

8.31.2.2 printResults()

```
void StatusBar::printResults (
    std::list< std::pair< parameterCombination, std::pair< functionValue, std::pair<
    ::filesystem::path >>> top ) [static]
```

Prints the given parameterCombinations and respective values to command line.

Parameters

<code>top</code>	List of parameterCombinations and respective values to be printed.
------------------	--

Definition at line 75 of file `StatusBar.cpp`.

References `LARGE_DIVIDER`, `printResult()`, and `SMALL_DIVIDER`.

8.31.2.3 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 Status and whose state is being printed.
---------------	--

Definition at line 70 of file `StatusBar.cpp`.

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

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

8.31.2.4 updateStatus()

```
void StatusBar::updateStatus (
    Status * opt,
    Status * runner,
    Status * eval,
    const std::pair< parameterCombination, 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 Optimizer used in optimization.
<i>runner</i>	Pointer to SimulationRunner used in optimization.
<i>eval</i>	Pointer to Evaluation used in optimization.
<i>currentVal</i>	<code>parameterCombination</code> 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 17 of file `StatusBar.cpp`.

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

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

8.31.3 Member Data Documentation

8.31.3.1 LARGE_DIVIDER

```
const std::string StatusBar::LARGE_DIVIDER = "\n\n" + std::string(70, '#') + "\n" [static],
[private]
```

Large divider used to visibly divide two sections of content.

Definition at line 43 of file `StatusBar.h`.

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

8.31.3.2 lastStatus

```
std::string StatusBar::lastStatus [private]
```

Last values of the [StatusBar](#) output (excluding value returned by [Status::getStatusBar](#))

Definition at line 60 of file `StatusBar.h`.

Referenced by `updateStatus()`.

8.31.3.3 lastStep

```
step StatusBar::lastStep = INIT [private]
```

Step which the optimization was in when `updateStatus` was called the last time.

Definition at line 56 of file `StatusBar.h`.

Referenced by `updateStatus()`.

8.31.3.4 lastVal

```
std::pair<parameterCombination, functionValue> StatusBar::lastVal [private]
```

Pair of `parameterCombination` and respective value used to discern if the best value has changed since the last call to `updateStatus`.

Definition at line 52 of file `StatusBar.h`.

Referenced by `updateStatus()`.

8.31.3.5 SMALL_DIVIDER

```
const std::string StatusBar::SMALL_DIVIDER = std::string(70, '-') + "\n" [static], [private]
```

Small divider used to visibly divide two sections of content.

Definition at line 47 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`

8.32 Controller::stepstate Struct Reference

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

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

Collaboration diagram for `Controller::stepstate`:

Controller::stepstate

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.

8.32.1 Detailed Description

A struct keeping track of the currently running optimization step for `StatusBar::updateStatus`.
Definition at line 124 of file `Controller.h`.

8.32.2 Member Function Documentation

8.32.2.1 `get()`

```
step Controller::stepstate::get ( ) [inline]
```

Returns the value of `currentStep`.

Returns

The step that is currently run.

Definition at line 146 of file `Controller.h`.

References `currentStep`, and `stepChanged`.

Referenced by `Controller::updateStatus()`, and `StubController::updateStatus()`.

8.32.2.2 `next()`

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

Switches `currentStep` to the next step.

Definition at line 137 of file `Controller.h`.

References `currentStep`, and `stepChanged`.

Referenced by `Controller::requestValues()`, and `Controller::run()`.

8.32.3 Member Data Documentation

8.32.3.1 `currentStep`

```
step Controller::stepstate::currentStep = INIT
```

Current step the optimization is in.

Definition at line 132 of file `Controller.h`.

Referenced by `get()`, and `next()`.

8.32.3.2 `stepChanged`

```
bool Controller::stepstate::stepChanged
```

Defines if `currentStep` has changed since the last call to `get`.

Definition at line 128 of file `Controller.h`.

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

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

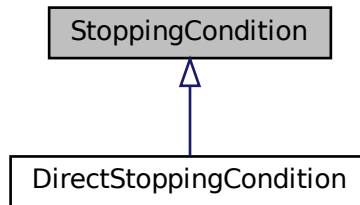
- `/home/runner/work/Simopticon/Simopticon/src/controller/Controller.h`

8.33 StoppingCondition Class Reference

A class used for deciding whether the optimization should be stopped.

```
#include "StoppingCondition.h"
```

Inheritance diagram for StoppingCondition:



Collaboration diagram for StoppingCondition:



Public Member Functions

- [StoppingCondition](#) (size_t evaluations=0, unsigned int minutes=0, [functionValue](#) accuracy=0, unsigned int accuracyIterations=0)
Creates a [StoppingCondition](#) with the given condition values.
- [StoppingCondition](#) (nlohmann::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, [functionValue](#) newBestVal)
Checks if any of the configured conditions is met for the given parameters.
- unsigned int [getIterationsSinceImprov](#) () const
Returns the value of [iterationsSinceImprov](#).

Static Protected Member Functions

- template<typename T >
static T [getConditionFromJSON](#) (nlohmann::json object, const std::string &key, const std::string &val="n")
Helper method, which checks whether the given condition should be used and returns the corresponding value if that's the case.

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.
- std::chrono::time_point< std::chrono::system_clock, std::chrono::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.

8.33.1 Detailed Description

A class used for deciding whether the optimization 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.

Author

Per Natzschka

Definition at line 21 of file StoppingCondition.h.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 StoppingCondition() [1/2]

```
StoppingCondition::StoppingCondition (
    size_t evaluations = 0,
    unsigned int minutes = 0,
    functionValue accuracy = 0,
    unsigned int accuracyIterations = 0 ) [explicit]
```

Creates a `StoppingCondition` with the given condition values.

Parameters

<code>evaluations</code>	Number of evaluations after which the optimization should stop.
<code>minutes</code>	Number of minutes after which the optimization should stop.
<code>accuracy</code>	Accuracy used in accuracy condition (see <code>ACCURACY</code>).
<code>accuracyIterations</code>	Number of iterations used in accuracy condition (see <code>NR_ACCURACY_ITERATIONS</code>).

Definition at line 9 of file StoppingCondition.cpp.

References ACCURACY, mins, NR_ACCURACY_ITERATIONS, NR_EVALUATIONS, and time_eval.

Referenced by DirectStoppingCondition::DirectStoppingCondition().

8.33.2.2 StoppingCondition() [2/2]

```
StoppingCondition::StoppingCondition (
    nlohmann::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 16 of file StoppingCondition.cpp.

References StoppingCondition().

Referenced by StoppingCondition().

8.33.3 Member Function Documentation

8.33.3.1 evaluate()

```
bool StoppingCondition::evaluate (
    size_t evaluations,
    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>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, and updateAccuracy().

Referenced by DirectStoppingCondition::evaluate().

8.33.3.2 getConditionFromJSON()

```
template<typename T >
static T StoppingCondition::getConditionFromJSON (
    nlohmann::json object,
    const std::string & key,
    const std::string & val = "n" ) [inline], [static], [protected]
```

Helper method, which checks whether the given condition should be used and returns the corresponding value if that's the case.

If not, 0 is returned.

Template Parameters

<i>T</i>	Type of the value located at the given key.
----------	---

Parameters

<i>object</i>	JSON object the condition is read from.
<i>key</i>	Key of the fetched condition.
<i>val</i>	Key of the value field of the fetched condition.

Returns

A value of type T that should be used as value for the condition.

Definition at line 83 of file StoppingCondition.h.

8.33.3.3 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](#).

8.33.3.4 setStartNow()

```
void StoppingCondition::setStartNow ( )
```

Sets [END_TIME](#) to be the current time plus [mins](#).
Definition at line 32 of file StoppingCondition.cpp.
References [time_eval](#).

8.33.3.5 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

<i>newBestVal</i>	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\(\)](#).

8.33.4 Member Data Documentation

8.33.4.1 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 46 of file `StoppingCondition.h`.

Referenced by `StoppingCondition()`, `evaluate()`, and `updateAccuracy()`.

8.33.4.2 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 56 of file `StoppingCondition.h`.

Referenced by `updateAccuracy()`.

8.33.4.3 END_TIME

```
std::chrono::time_point<std::chrono::system_clock, std::chrono::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 32 of file `StoppingCondition.h`.

8.33.4.4 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 61 of file `StoppingCondition.h`.

Referenced by `getIterationsSinceImprov()`, and `updateAccuracy()`.

8.33.4.5 mins

```
const unsigned int StoppingCondition::mins [private]
```

Number of minutes after which the optimization should stop.

Definition at line 36 of file `StoppingCondition.h`.

Referenced by `StoppingCondition()`.

8.33.4.6 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 51 of file `StoppingCondition.h`.

Referenced by `StoppingCondition()`, `evaluate()`, and `updateAccuracy()`.

8.33.4.7 NR_EVALUATIONS

```
const size_t StoppingCondition::NR_EVALUATIONS [private]
```

Number of evaluations after which the optimization should stop.
Definition at line 26 of file StoppingCondition.h.
Referenced by StoppingCondition(), and evaluate().

8.33.4.8 time_eval

```
bool StoppingCondition::time_eval [private]
```

Defines whether the time condition should be used.

Definition at line 40 of file StoppingCondition.h.

Referenced by StoppingCondition(), and setStartNow().

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

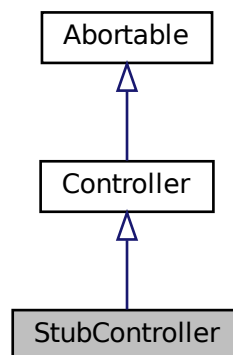
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.cpp](#)

8.34 StubController Class Reference

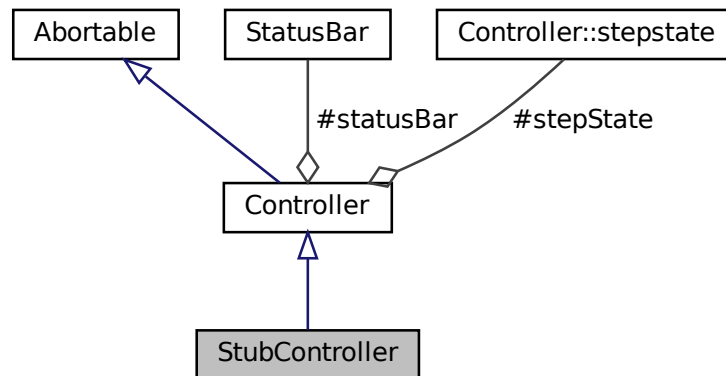
A class that mocks behaviour of [Controller](#).

```
#include "StubController.h"
```

Inheritance diagram for StubController:



Collaboration diagram for StubController:



Public Member Functions

- [StubController](#) (const std::filesystem::path &configPath, const std::string &function)
Creates a [StubController](#) with the given config and function.

Private Member Functions

- std::map< [parameterCombination](#), std::pair< std::filesystem::path, std::set< [runId](#) > >, [CmpVectorSharedParameter](#) > [runSimulations](#) (const std::set< [parameterCombination](#), [CmpVectorSharedParameter](#) > &runs) override
Returns empty paths and runIds for each requested parameterCombination.
- std::map< [parameterCombination](#), [functionValue](#), [CmpVectorSharedParameter](#) > [evaluate](#) (const std::map< [parameterCombination](#), std::pair< std::filesystem::path, std::set< [runId](#) > >, [CmpVectorSharedParameter](#) > &simulationResults) override
Evaluates the given parameterCombinations 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 std::function< [functionValue](#)([parameterCombination](#))> f
Function to be optimized in the current optimization.

Static Private Attributes

- static std::map< std::string, std::function< [functionValue](#)([parameterCombination](#))> > functions
Map that contains the predefined functions [quadratic](#), [shekel5](#), [shekel7](#), [shekel10](#), [branin](#), [goldprice](#), [camel6](#), [shubert](#), [hartman3](#) and [hartman6](#).

Additional Inherited Members

8.34.1 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>

Author

Per Natzschka

Definition at line 22 of file StubController.h.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 StubController()

```
StubController::StubController (
    const std::filesystem::path & configPath,
    const std::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 143 of file StubController.cpp.

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

8.34.3 Member Function Documentation

8.34.3.1 evaluate()

```
std::map< parameterCombination, functionValue, CmpVectorSharedParameter > StubController←
::evaluate (
    const std::map< parameterCombination, std::pair< std::filesystem::path, std←
::set< runId >>, CmpVectorSharedParameter > & simulationResults ) [override], [private],
[virtual]
```

Evaluates the given parameterCombinations with [f](#).

Parameters

<i>simulationResults</i>	Map which maps parameterCombinations to empty results (see runSimulations).
--------------------------	--

Returns

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

Reimplemented from [Controller](#).

Definition at line 156 of file StubController.cpp.

References [f](#).

8.34.3.2 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 165 of file StubController.cpp.

8.34.3.3 runSimulations()

```
std::map< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, CmpVectorSharedParameter
> StubController::runSimulations (
    const std::set< parameterCombination, CmpVectorSharedParameter > & runs ) [override],
[private], [virtual]
```

Returns empty paths and runlds for each requested parameterCombination.

Parameters

<i>runs</i>	parameterCombination to be simulated.
-------------	---------------------------------------

Returns

Map which maps the given parameterCombinations to empty paths and runlds.

Reimplemented from [Controller](#).
Definition at line 148 of file StubController.cpp.

8.34.3.4 updateStatus()

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

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

Reimplemented from [Controller](#).

Definition at line 168 of file StubController.cpp.

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

8.34.4 Member Data Documentation

8.34.4.1 f

```
const std::function<functionValue(parameterCombination)> StubController::f [private]
```

Function to be optimized in the current optimization.

One of the functions in [functions](#).

Definition at line 33 of file StubController.h.

Referenced by [StubController\(\)](#), and [evaluate\(\)](#).

8.34.4.2 functions

```
std::map< std::string, std::function< functionValue(parameterCombination)> > StubController←
::functions [static], [private]
```

Map that contains the predefined functions quadratic, shekel5, shekel7, shekel10, branin, goldprice, camel6, shu-
bert, hartman3 and hartman6.

For more information on all but the first function visit: <https://www.sfu.ca/~ssurjano/optimization.html>

Definition at line 28 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](#)

8.35 ThreadsafeQueue< Key > Class Template Reference

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

```
#include "ThreadsafeQueue.h"
```

Collaboration diagram for ThreadsafeQueue< Key >:

ThreadsafeQueue< Key >

Public Member Functions

- void [push](#) (Key val)
Adds the given value to [safeQueue](#).
- std::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

- std::queue< Key > [safeQueue](#)
The actual queue data structure.
- std::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.

8.35.1 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.
------------	--

Author

Per Natzschka

Definition at line 19 of file ThreadsafeQueue.h.

8.35.2 Member Function Documentation

8.35.2.1 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.

Definition at line 39 of file ThreadsafeQueue.hpp.

References ThreadsafeQueue< Key >::queueLock, and ThreadsafeQueue< Key >::safeQueue.

8.35.2.2 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.

Definition at line 31 of file ThreadsafeQueue.hpp.

References ThreadsafeQueue< Key >::queueLock, and ThreadsafeQueue< Key >::startSize.

8.35.2.3 pop()

```
template<class Key >
std::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.

Definition at line 18 of file ThreadsafeQueue.hpp.

References ThreadsafeQueue< Key >::queueLock, and ThreadsafeQueue< Key >::safeQueue.

8.35.2.4 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.
------------	------------------------------

Definition at line 10 of file ThreadsafeQueue.hpp.

References ThreadsafeQueue< Key >::queueLock, ThreadsafeQueue< Key >::safeQueue, and ThreadsafeQueue< Key >::startSize.

8.35.3 Member Data Documentation

8.35.3.1 queueLock

```
template<class Key >
```

```
std::mutex ThreadsafeQueue< Key >::queueLock [private]
```

Threadlock to avoid damage to [safeQueue](#) on concurrent access.

Definition at line 28 of file ThreadsafeQueue.h.

Referenced by ThreadsafeQueue< Key >::getSize(), ThreadsafeQueue< Key >::getStartSize(), ThreadsafeQueue< Key >::pop(), and ThreadsafeQueue< Key >::push().

8.35.3.2 safeQueue

```
template<class Key >
```

```
std::queue<Key> ThreadsafeQueue< Key >::safeQueue [private]
```

The actual queue data structure.

Definition at line 24 of file ThreadsafeQueue.h.

Referenced by ThreadsafeQueue< Key >::getSize(), ThreadsafeQueue< Key >::pop(), and ThreadsafeQueue< Key >::push().

8.35.3.3 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 33 of file ThreadsafeQueue.h.

Referenced by ThreadsafeQueue< Key >::getStartSize(), and ThreadsafeQueue< Key >::push().

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

- [/home/runner/work/Simopticon/Simopticon/src/utils/ThreadsafeQueue.h](#)
- [/home/runner/work/Simopticon/Simopticon/src/utils/ThreadsafeQueue.hpp](#)

8.36 ValueMap Class Reference

A container managing a map data structure that maps parameterCombinations to their respective found values.

```
#include "ValueMap.h"
```

Collaboration diagram for ValueMap:

```

classDiagram
    class ValueMap
  
```

Public Member Functions

- [ValueMap](#) (unsigned int [topEntries](#)=10)
Creates a [ValueMap](#).
- [functionValue query](#) (const [parameterCombination](#) ¶ms)
Returns the value saved at the given [parameterCombination](#).
- void [insert](#) (const [parameterCombination](#) ¶ms, [functionValue](#) val)
Adds the given [parameterCombination](#) and value to [tba](#).

- bool `isKnown` (const `parameterCombination` &cords)
Checks if a value has been recorded at the given parameterCombination.
- bool `isTopValue` (const `parameterCombination` &cords)
Checks if the given parameterCombination is to be found in `topVals`.
- const `std::map`< `parameterCombination`, `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.
- `std::list`< `std::pair`< `parameterCombination`, `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 `std::pair`< `parameterCombination`, `functionValue` > &val, `std::set`< `functionValue` *, `CmpPtrFunctionvalue` > &set)
Inserts a single value into `values` and into `lowerValues` or `upperValues` depending on set argument.

Private Attributes

- `std::mutex` `operationsLock`
Threadlock to avoid damage to the data structure when concurrent threads access it.
- `std::set`< `functionValue` *, `CmpPtrFunctionvalue` > `upperValues`
Greater half of the values in `values`.
- `std::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.
- `std::set`< `std::pair`< const `parameterCombination`, `functionValue` >, `CmpPairVectorSharedParameterFunctionvalue` > `topVals`
Set of pairs of the best parameterCombinations and their respective values.
- `std::map`< `parameterCombination`, `functionValue`, `CmpVectorSharedParameter` > `values`
Actual map that contains parameterCombinations and their respective values.
- `std::list`< `std::pair`< `parameterCombination`, `functionValue` > > `tba`
Entries that have been added since last `updateMap`.

8.36.1 Detailed Description

A container managing a map data structure that maps `parameterCombinations` 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.

Author

Per Natzschka

Definition at line 29 of file `ValueMap.h`.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 ValueMap()

```
ValueMap::ValueMap (
    unsigned int topEntries = 10 ) [explicit]
```

Creates a [ValueMap](#).

Parameters

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

Definition at line 13 of file ValueMap.cpp.

References [topEntries](#).

8.36.3 Member Function Documentation

8.36.3.1 addValue()

```
void ValueMap::addValue (
    const std::pair< parameterCombination, functionValue > & val,
    std::set< functionValue *, CmpPtrFunctionvalue > & set ) [private]
```

Inserts a single value into [values](#) and into [lowerValues](#) or [upperValues](#) depending on *set* argument.

Parameters

<i>val</i>	parameterCombination and respective value to be inserted.
<i>set</i>	Set that value is inserted in. Either lowerValues or upperValues .

Definition at line 51 of file ValueMap.cpp.

References [topEntries](#), [topVals](#), and [values](#).

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

8.36.3.2 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 98 of file ValueMap.cpp.

References [getSize\(\)](#), [lowerValues](#), [operationsLock](#), [updateMap\(\)](#), and [upperValues](#).

8.36.3.3 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 110 of file ValueMap.cpp.

References [tba](#), and [values](#).

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

8.36.3.4 getTopVals()

```
std::list< std::pair< parameterCombination, functionValue > > ValueMap::getTopVals ( )
```

Returns the best [topEntries](#) entries that are saved in [topVals](#).

Triggers [updateMap](#).

Returns

A list of the best [topEntries](#) parameterCombinations and their respective values.

Definition at line 114 of file ValueMap.cpp.

References [topVals](#), and [updateMap\(\)](#).

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

8.36.3.5 getValues()

```
const std::map< parameterCombination, functionValue, CmpVectorSharedParameter > & ValueMap←  
::getValues ( )
```

Returns the whole [values](#) member.

Triggers [updateMap](#).

Returns

A map reference to [values](#).

Definition at line 139 of file ValueMap.cpp.

References [updateMap\(\)](#), and [values](#).

8.36.3.6 insert()

```
void ValueMap::insert (
    const parameterCombination & params,
    functionValue val )
```

Adds the given parameterCombination and value to [tba](#).

Parameters

<i>params</i>	parameterCombination to be added.
<i>val</i>	Value to be added.

Definition at line 83 of file ValueMap.cpp.

References [tba](#).

8.36.3.7 isKnown()

```
bool ValueMap::isKnown (
    const parameterCombination & cords )
```

Checks if a value has been recorded at the given parameterCombination.

Triggers [updateMap](#).

Parameters

<i>cords</i>	parameterCombination that is checked.
--------------	---------------------------------------

Returns

A boolean value that represents if the value is known.

Definition at line 87 of file ValueMap.cpp.

References `getSize()`, `operationsLock`, `updateMap()`, and `values`.

8.36.3.8 isTopValue()

```
bool ValueMap::isTopValue (
    const parameterCombination & cords )
```

Checks if the given parameterCombination is to be found in `topVals`.

Triggers `updateMap`.

Parameters

<i>cords</i>	parameterCombination that is checked.
--------------	---------------------------------------

Returns

A boolean value that represents if the value is one of the best `topEntries` entries in `values`.

Definition at line 119 of file ValueMap.cpp.

References `Parameter::getVal()`, `topVals`, and `updateMap()`.

8.36.3.9 query()

```
functionValue ValueMap::query (
    const parameterCombination & params )
```

Returns the value saved at the given parameterCombination.

If no value is present, an exception is thrown. Triggers `updateMap`.

Parameters

<i>params</i>	parameterCombination to which the value is requested.
---------------	---

Returns

The value saved in `values` at the given parameterCombination.

Definition at line 72 of file ValueMap.cpp.

References `operationsLock`, `updateMap()`, and `values`.

8.36.3.10 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 16 of file ValueMap.cpp.

References `addValue()`, `lowerValues`, `operationsLock`, `tba`, and `upperValues`.

Referenced by `getMedian()`, `getTopVals()`, `getValues()`, `isKnown()`, `isTopValue()`, and `query()`.

8.36.4 Member Data Documentation

8.36.4.1 lowerValues

```
std::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 43 of file ValueMap.h.

Referenced by [getMedian\(\)](#), and [updateMap\(\)](#).

8.36.4.2 operationsLock

```
std::mutex ValueMap::operationsLock [private]
```

Threadlock to avoid damage to the data structure when concurrent threads access it.

Definition at line 34 of file ValueMap.h.

Referenced by [getMedian\(\)](#), [isKnown\(\)](#), [query\(\)](#), and [updateMap\(\)](#).

8.36.4.3 tba

```
std::list<std::pair<parameterCombination, 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 65 of file ValueMap.h.

Referenced by [getSize\(\)](#), [insert\(\)](#), and [updateMap\(\)](#).

8.36.4.4 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 49 of file ValueMap.h.

Referenced by [ValueMap\(\)](#), and [addValue\(\)](#).

8.36.4.5 topVals

```
std::set<std::pair<const parameterCombination, functionValue>, CmpPairVectorSharedParameterFunctionvalue>
ValueMap::topVals [private]
```

Set of pairs of the best parameterCombinations and their respective values.

Contains not more than [topEntries](#) entries.

Definition at line 54 of file ValueMap.h.

Referenced by [addValue\(\)](#), [getTopVals\(\)](#), and [isTopValue\(\)](#).

8.36.4.6 upperValues

```
std::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 39 of file ValueMap.h.

Referenced by [getMedian\(\)](#), and [updateMap\(\)](#).

8.36.4.7 values

```
std::map<parameterCombination, functionValue, CmpVectorSharedParameter> ValueMap::values
[private]
```

Actual map that contains parameterCombinations and their respective values.

Definition at line 59 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 9

File Documentation

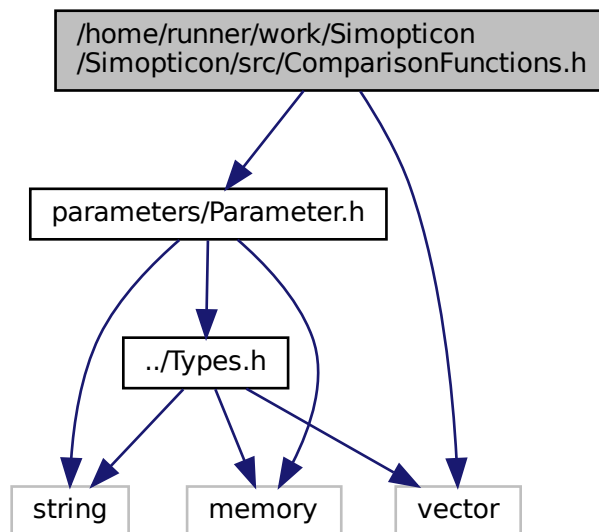
9.1 `/home/runner/work/Simopticon/Simopticon/src/ComparisonFunctions.h` File Reference

In this file, comparison functions are defined which should be used across the whole framework.

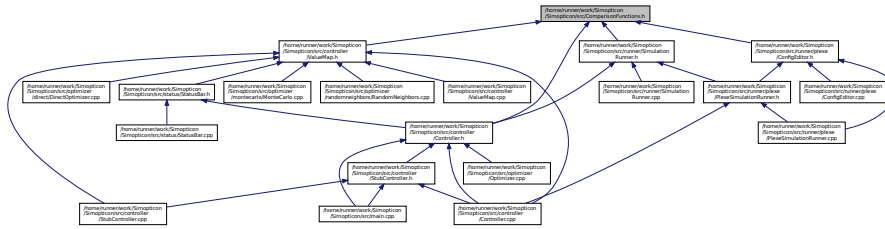
```
#include "parameters/Parameter.h"
```

```
#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 parameterCombination and function value.

9.1.1 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.

Author

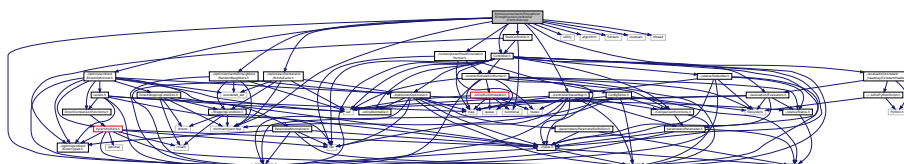
Per Natzschka

9.2 /home/runner/work/Simopticon/Simopticon/src/controller/↵ Controller.cpp File Reference

In this file, the implementation of the [Controller](#) class is defined.

```
#include "Controller.h"
#include <memory>
#include <utility>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <thread>
#include "StubController.h"
#include "ValueMap.h"
#include "../optimizer/direct/DirectOptimizer.h"
#include "../optimizer/montecarlo/MonteCarlo.h"
#include "../optimizer/randomneighbors/RandomNeighbors.h"
#include "../runner/plex/PluxeSimulationRunner.h"
#include "../evaluation/constant_headway/ConstantHeadway.h"
#include "nlohmann/json.hpp"
```

Include dependency graph for Controller.cpp:



Functions

- `nlohmann::json getConfigByPath` (`const std::filesystem::path &baseDir`, `const std::string &config`)
Helper method parsing a json object from the given file.

9.2.1 Detailed Description

In this file, the implementation of the `Controller` class is defined.

Author

Per Natzschka

9.2.2 Function Documentation

9.2.2.1 getConfigByPath()

```
nlohmann::json getConfigByPath (
    const std::filesystem::path & baseDir,
    const std::string & config )
```

Helper method parsing a json object from the given file.

Parameters

<i>baseDir</i>	Directory the json file resides in.
<i>config</i>	Name of the json file.

Returns

A json object parsed from the given file.

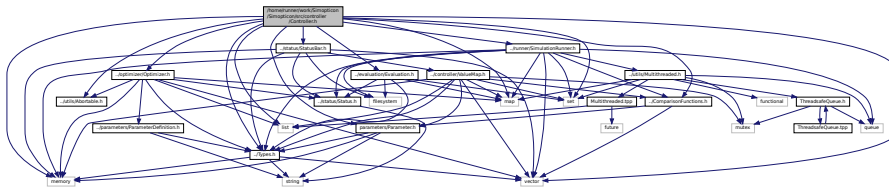
Definition at line 31 of file `Controller.cpp`.

9.3 /home/runner/work/Simopticon/Simopticon/src/controller/↵ Controller.h File Reference

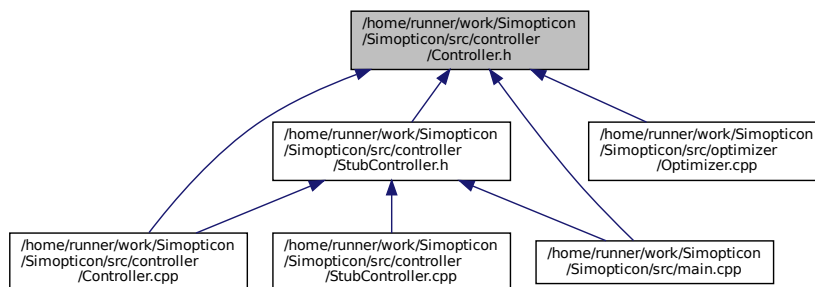
In this file, the header of the `Controller` class is defined.

```
#include "../Types.h"
#include "../ComparisonFunctions.h"
#include "../optimizer/Optimizer.h"
#include "../runner/SimulationRunner.h"
#include "../evaluation/Evaluation.h"
#include "../parameters/Parameter.h"
#include "../status/StatusBar.h"
#include "../utils/Abortable.h"
#include <map>
#include <vector>
#include <list>
#include <memory>
#include <set>
```

Include dependency graph for Controller.h:



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



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](#).

9.3.1 Detailed Description

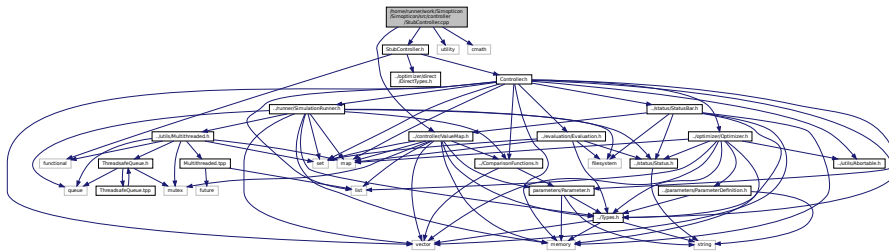
In this file, the header of the [Controller](#) class is defined.

9.4 [/home/runner/work/Simopticon/Simopticon/src/controller/StubController.cpp](#) File Reference

In this file, the implementation of the [StubController](#) class is defined.

```
#include "StubController.h"
#include "ValueMap.h"
#include <utility>
#include <cmath>
```

Include dependency graph for StubController.cpp:



Functions

- `functionValue shekel` (int m, const `parameterCombination` &v)
Helper method calculating the Shekel function with m local minima for the given input.
- `functionValue hartman` (const std::vector< std::vector< `functionValue` >> &A, const std::vector< std::vector< `functionValue` >> &P, const `parameterCombination` &v)
Helper method calculating the Hartman function with the given A and P matrices for the given input.

9.4.1 Detailed Description

In this file, the implementation of the `StubController` class is defined.

Author

Per Natzschka

9.4.2 Function Documentation

9.4.2.1 hartman()

```
functionValue hartman (
    const std::vector< std::vector< functionValue >> & A,
    const std::vector< std::vector< functionValue >> & P,
    const parameterCombination & v )
```

Helper method calculating the Hartman function with the given *A* and *P* matrices for the given input.

More information at <https://www.sfu.ca/~ssurjano/hart3.html>, <https://www.sfu.ca/~ssurjano/hart4.html> and <https://www.sfu.ca/~ssurjano/hart6.html>.

Parameters

<i>A</i>	Matrix A used in calculation of Hartman function.
<i>P</i>	Matrix P used in calculation of Hartman function.
<i>v</i>	

Returns

Definition at line 48 of file StubController.cpp.

9.4.2.2 shekel()

```
functionValue shekel (
```

```
int m,
const parameterCombination & v )
```

Helper method calculating the Shekel function with m local minima for the given input.

Only implemented for $1 \leq m \leq 10$. More information at <https://www.sfu.ca/~ssurjano/shekel.html>.

Parameters

m	Number of local minima.
v	Argument vector where the function should be evaluated.

Returns

The value of the shekel function at the given argument vector.

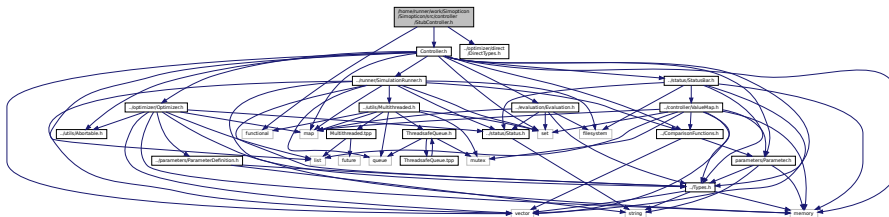
Definition at line 21 of file StubController.cpp.

9.5 /home/runner/work/Simopticon/Simopticon/src/controller/StubController.h File Reference

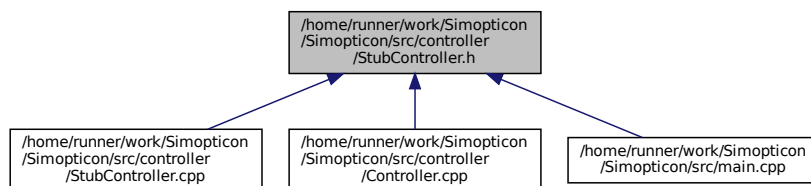
In this file, the header of the `StubController` class is defined.

```
#include <functional>
#include "Controller.h"
#include "../optimizer/direct/DirectTypes.h"
```

Include dependency graph for StubController.h:



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



Classes

- class `StubController`

A class that mocks behaviour of `Controller`.

9.5.1 Detailed Description

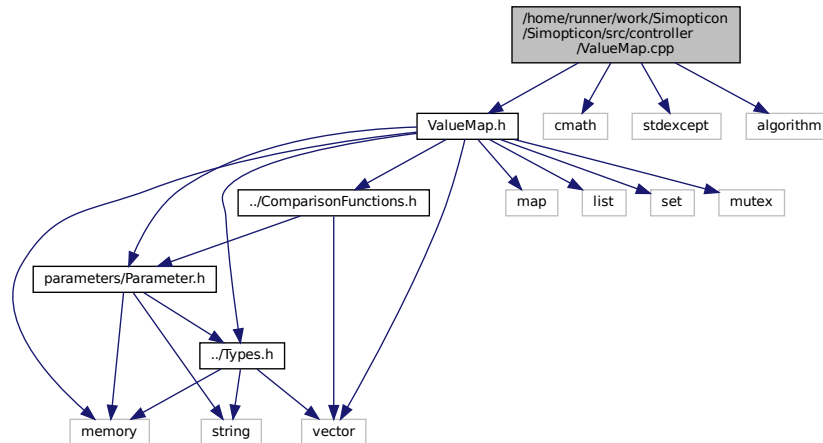
In this file, the header of the `StubController` class is defined.

9.6 /home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.cpp File Reference

In this file, the implementation of the [ValueMap](#) class is defined.

```
#include "ValueMap.h"
#include <cmath>
#include <stdexcept>
#include <algorithm>
```

Include dependency graph for ValueMap.cpp:



9.6.1 Detailed Description

In this file, the implementation of the [ValueMap](#) class is defined.

Author

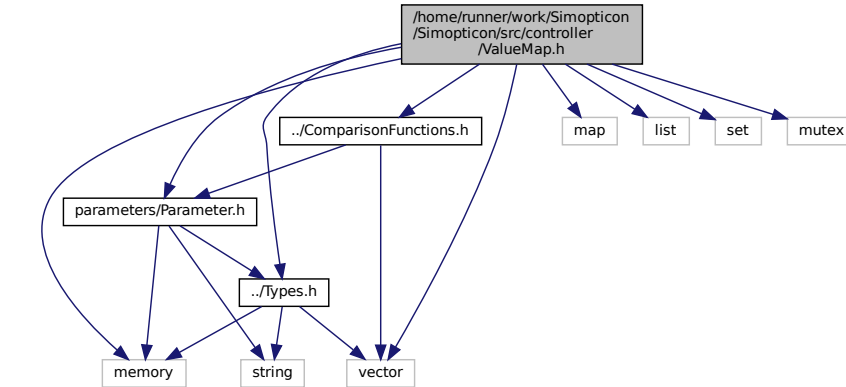
Per Natzschka

9.7 /home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.h File Reference

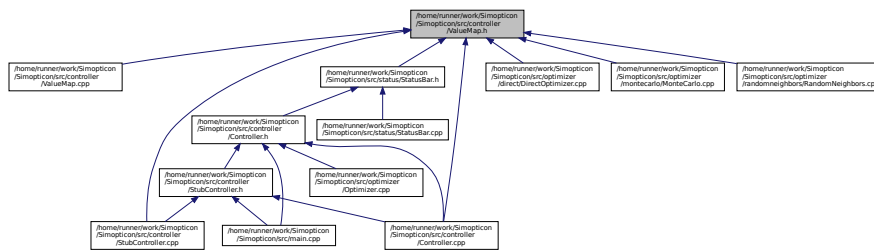
In this file, the header of the [ValueMap](#) class is defined.

```
#include "../Types.h"
#include "../ComparisonFunctions.h"
#include "../parameters/Parameter.h"
#include <map>
#include <vector>
#include <list>
#include <memory>
#include <set>
#include <mutex>
```

Include dependency graph for ValueMap.h:



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



Classes

- class [ValueMap](#)

A container managing a map data structure that maps parameterCombinations to their respective found values.

9.7.1 Detailed Description

In this file, the header of the [ValueMap](#) class is defined.

9.8 /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/constant_headway.py File Reference

In this file, Python functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap is defined.

Functions

- 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.

9.8.1 Detailed Description

In this file, Python functionality for automatic rating of Plexe result files on the mean deviation from the pre-defined gap is defined.

To achieve this, the OMNeT++ Python API `omnetpp.scave` is used. Multithreading is introduced to speed up the processing of multiple evaluations. Wrapped by [ConstantHeadway](#) class.

Author

Per Natzschka

9.8.2 Function Documentation

9.8.2.1 `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.

Definition at line 32 of file `constant_headway.py`.

9.8.2.2 `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 parameterCombination

Returns

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

Definition at line 70 of file constant_headway.py.

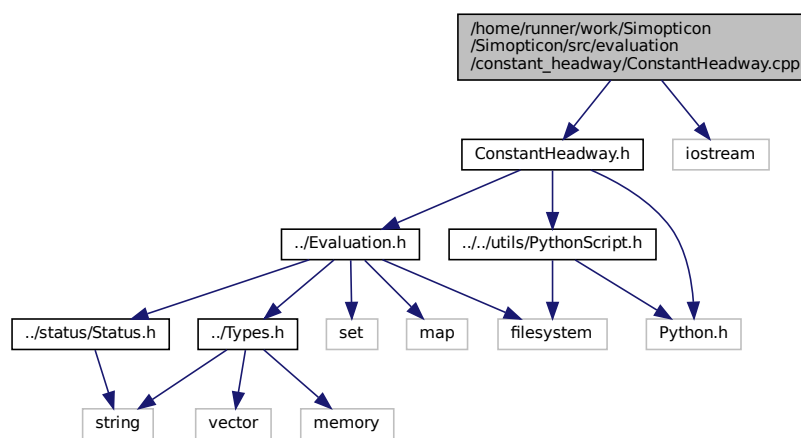
9.9 /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.cpp File Reference

In this file, the implementation of the [ConstantHeadway](#) class is defined.

```
#include "ConstantHeadway.h"
```

```
#include <iostream>
```

Include dependency graph for ConstantHeadway.cpp:



9.9.1 Detailed Description

In this file, the implementation of the [ConstantHeadway](#) class is defined.

Author

Per Natzschka

9.10 /home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.h File Reference

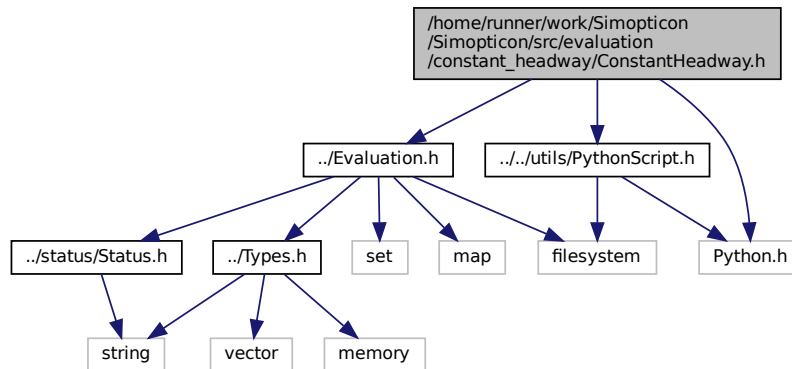
In this file, the header of the [ConstantHeadway](#) class is defined.

```
#include "../Evaluation.h"
```

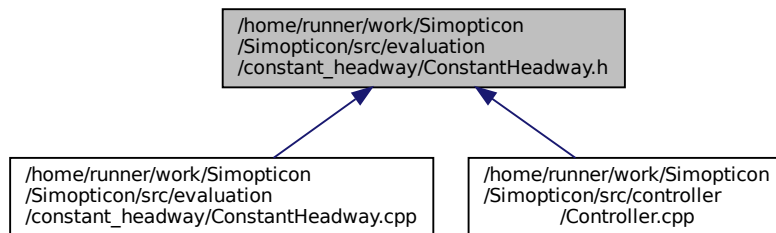
```
#include "../utils/PythonScript.h"
```

```
#include <Python.h>
```

Include dependency graph for ConstantHeadway.h:



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



Classes

- class [ConstantHeadway](#)
A wrapper for the [constant_headway.py](#) script.

9.10.1 Detailed Description

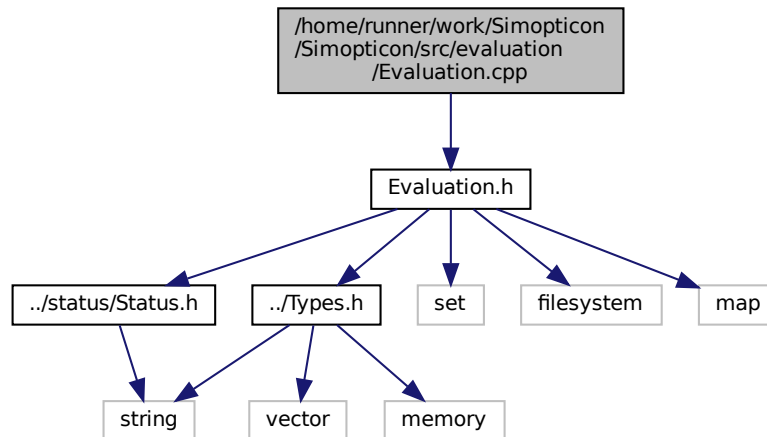
In this file, the header of the [ConstantHeadway](#) class is defined.

9.11 /home/runner/work/Simopticon/Simopticon/src/evaluation/↵ Evaluation.cpp File Reference

In this file, the implementation of the [Evaluation](#) class is defined.

```
#include "Evaluation.h"
```

Include dependency graph for Evaluation.cpp:



9.11.1 Detailed Description

In this file, the implementation of the [Evaluation](#) class is defined.

Author

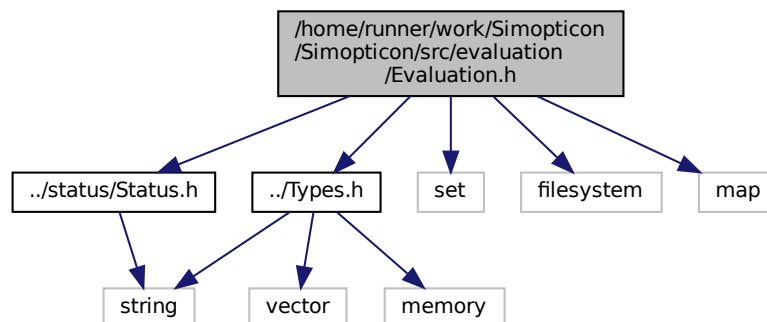
Per Natzschka

9.12 /home/runner/work/Simopticon/Simopticon/src/evaluation/↵ Evaluation.h File Reference

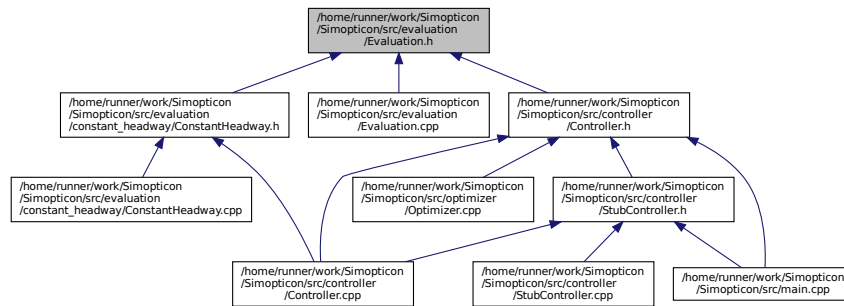
In this file, the header of the [Evaluation](#) class is defined.

```
#include "../Types.h"
#include "../status/Status.h"
#include <set>
#include <filesystem>
#include <map>
```

Include dependency graph for Evaluation.h:



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



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.

9.12.1 Detailed Description

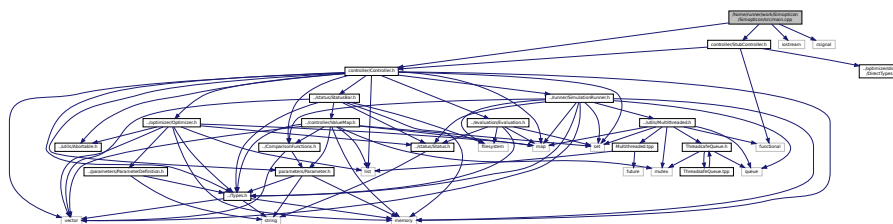
In this file, the header of the [Evaluation](#) class is defined.

9.13 /home/runner/work/Simopticon/Simopticon/src/main.cpp File Reference

In this file, the main function running the *Simopticon* framework is defined.

```
#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

- `std::unique_ptr< Controller > ctr`

Reference to the [Controller](#) that is running the optimization.

9.13.1 Detailed Description

In this file, the main function running the *Simopticon* framework is defined.

Author

Per Natzschka

9.13.2 Function Documentation

9.13.2.1 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

<code>s</code>	Necessary parameter for interrupt handlers (unused).
----------------	--

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

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

Definition at line 29 of file main.cpp.

9.13.2.2 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

<code>argc</code>	Number of command line arguments.
<code>argv</code>	Array of command line arguments.

Returns

Status code.

Definition at line 44 of file main.cpp.

9.13.3 Variable Documentation

9.13.3.1 ctr

```
std::unique_ptr< Controller > ctr
```

Reference to the [Controller](#) that is running the optimization.

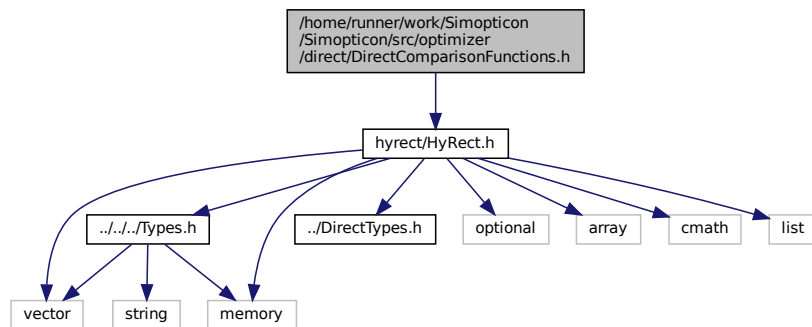
Definition at line 23 of file main.cpp.

9.14 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ DirectComparisonFunctions.h File Reference

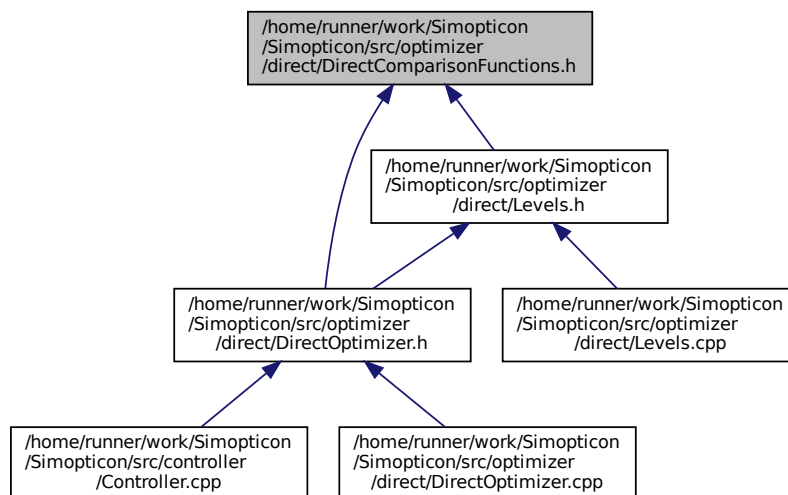
In this file, comparison functions are defined which are used in the direct module.

```
#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.

9.14.1 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.

Author

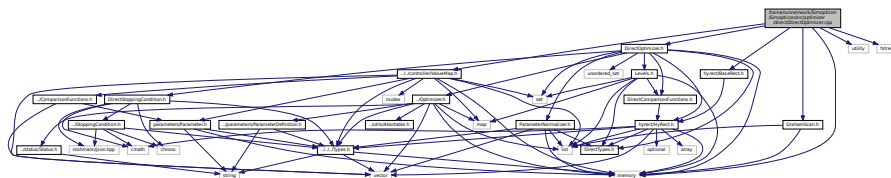
Per Natzschka

9.15 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ DirectOptimizer.cpp File Reference

In this file, the implementation of the [DirectOptimizer](#) class is defined.

```
#include "DirectOptimizer.h"
#include "GrahamScan.h"
#include "hyrect/BaseRect.h"
#include "../controller/ValueMap.h"
#include <utility>
#include <memory>
#include <fstream>
```

Include dependency graph for DirectOptimizer.cpp:



9.15.1 Detailed Description

In this file, the implementation of the [DirectOptimizer](#) class is defined.

Author

Per Natzschka

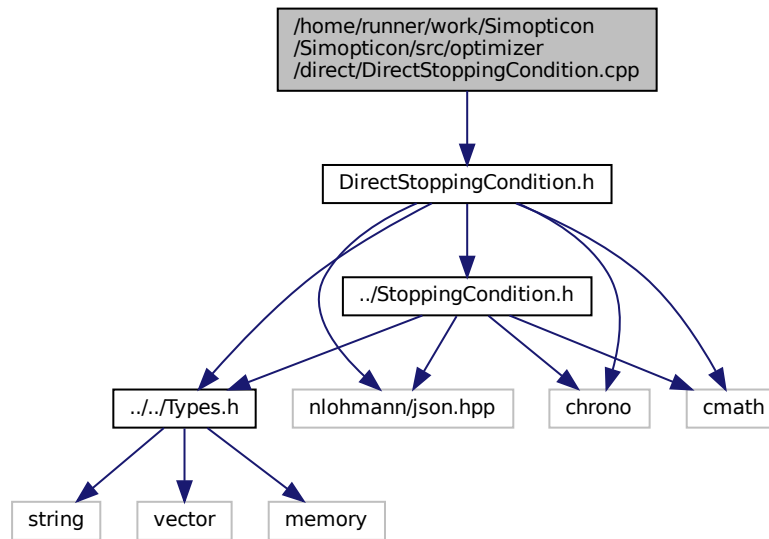
9.16 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ DirectOptimizer.h File Reference

In this file, the header of the [DirectOptimizer](#) class is defined.

```
#include "DirectTypes.h"
#include "DirectComparisonFunctions.h"
#include "../Optimizer.h"
#include "DirectStoppingCondition.h"
#include "hyrect/HyRect.h"
#include "ParameterNormalizer.h"
#include "Levels.h"
#include <set>
#include <unordered_set>
#include <memory>
```



```
#include "DirectStoppingCondition.h"
Include dependency graph for DirectStoppingCondition.cpp:
```



9.17.1 Detailed Description

In this file, the implementation of the [DirectStoppingCondition](#) class is defined.

Author

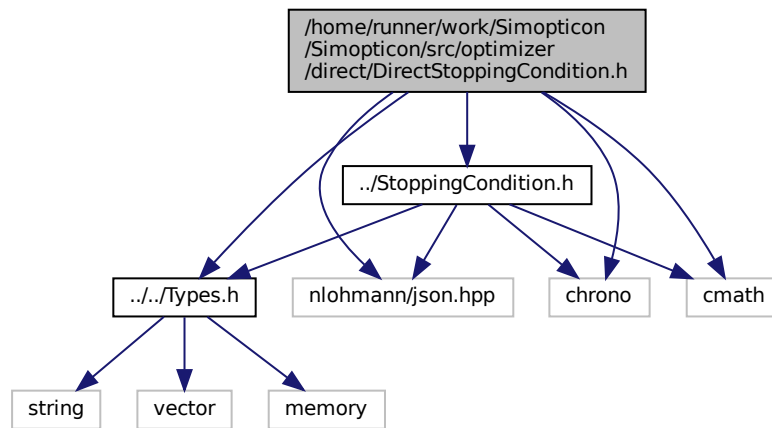
Per Natzschka

9.18 `/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/` `DirectStoppingCondition.h` File Reference

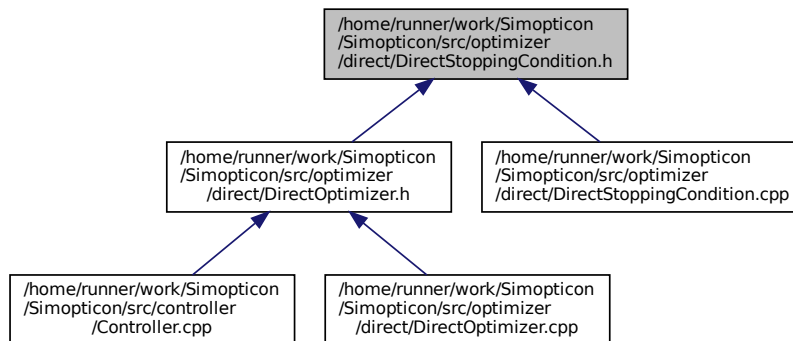
In this file, the header of the [StoppingCondition](#) class is defined.

```
#include "../../Types.h"
#include "nlohmann/json.hpp"
#include "../StoppingCondition.h"
#include <chrono>
#include <cmath>
```

Include dependency graph for DirectStoppingCondition.h:



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



Classes

- class [DirectStoppingCondition](#)
A class used for deciding whether the DIRECT should be stopped.

9.18.1 Detailed Description

In this file, the header of the [StoppingCondition](#) class is defined.

9.19 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ DirectTypes.h File Reference

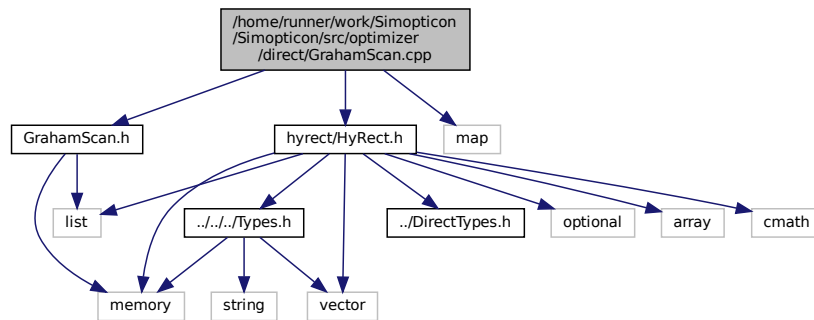
In this file, types are defined which are used in the direct module.

9.20 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ GrahamScan.cpp File Reference

In this file, the implementation of the [GrahamScan](#) class is defined.

```
#include "GrahamScan.h"
#include "hyrect/HyRect.h"
#include <map>
```

Include dependency graph for GrahamScan.cpp:



9.20.1 Detailed Description

In this file, the implementation of the [GrahamScan](#) class is defined.

Author

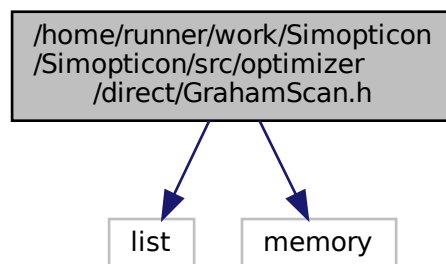
Per Natzschka

9.21 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ GrahamScan.h File Reference

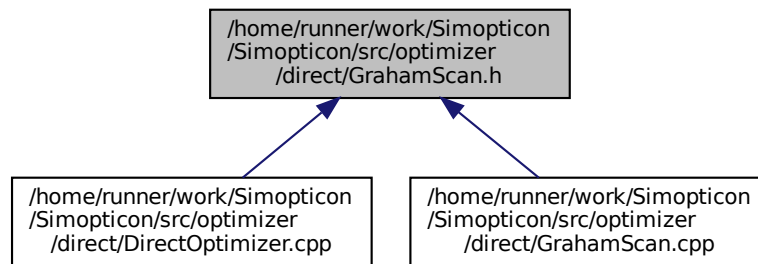
In this file, the header of the [GrahamScan](#) class is defined.

```
#include <list>
#include <memory>
```

Include dependency graph for GrahamScan.h:



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



Classes

- class [GrahamScan](#)

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

9.21.1 Detailed Description

In this file, the header of the [GrahamScan](#) class is defined.

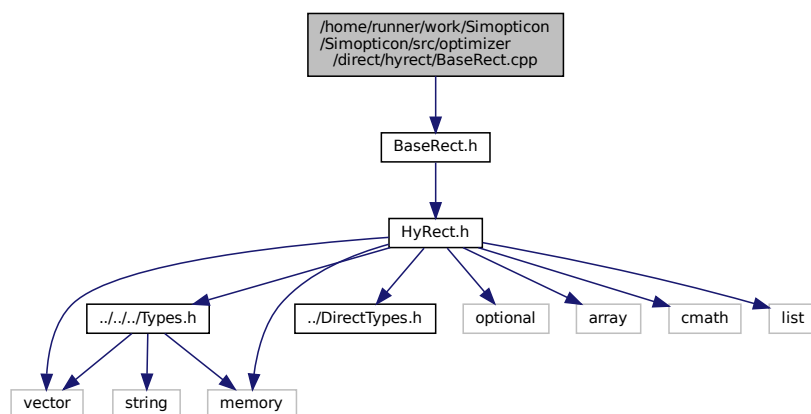
9.22 /home/runner/work/Simopticon/↵

Simopticon/src/optimizer/direct/hyrect/BaseRect.cpp File Reference

In this file, the implementation of the [BaseRect](#) class is defined.

```
#include "BaseRect.h"
```

Include dependency graph for BaseRect.cpp:



9.22.1 Detailed Description

In this file, the implementation of the [BaseRect](#) class is defined.

Author

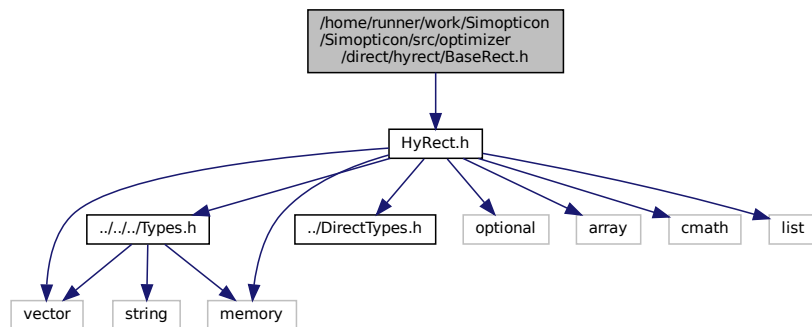
Per Natzschka

9.23 /home/runner/work/Simopticon/↔ Simopticon/src/optimizer/direct/hyrect/BaseRect.h File Reference

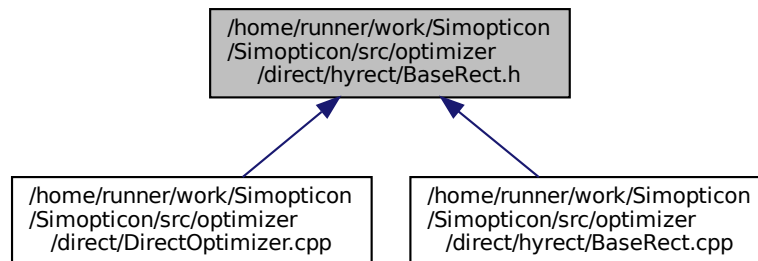
In this file, the header of the [BaseRect](#) class is defined.

```
#include "HyRect.h"
```

Include dependency graph for BaseRect.h:



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



Classes

- class [BaseRect](#)

A class representing a [HyRect](#) without a parent rectangle.

9.23.1 Detailed Description

In this file, the header of the [BaseRect](#) class is defined.

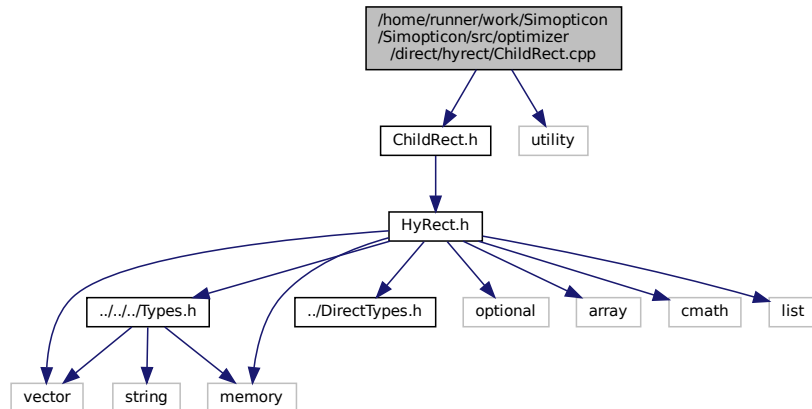
9.24 /home/runner/work/Simopticon/↵ Simopticon/src/optimizer/direct/hyrect/ChildRect.cpp File Reference

In this file, the implementation of the [ChildRect](#) class is defined.

```
#include "ChildRect.h"
```

```
#include <utility>
```

Include dependency graph for ChildRect.cpp:



9.24.1 Detailed Description

In this file, the implementation of the [ChildRect](#) class is defined.

Author

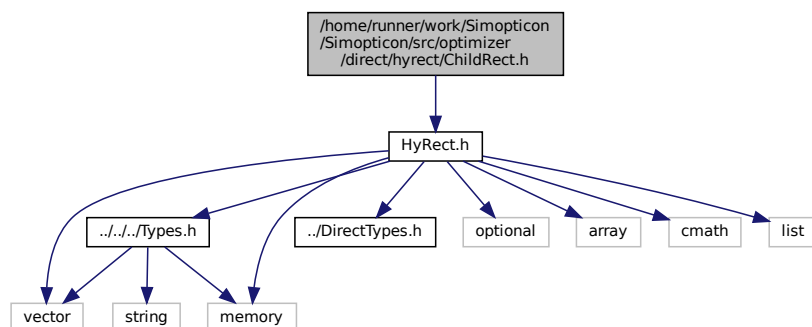
Per Natzschka

9.25 /home/runner/work/Simopticon/↵ Simopticon/src/optimizer/direct/hyrect/ChildRect.h File Reference

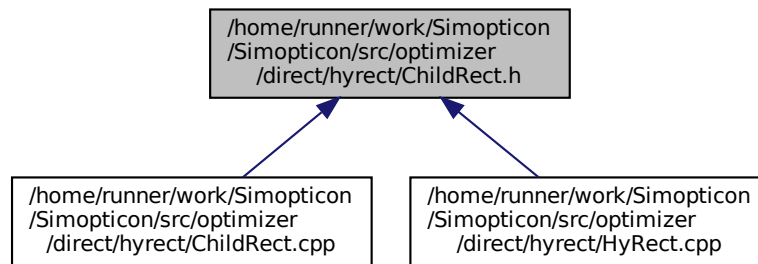
In this file, the header of the [ChildRect](#) class is defined.

```
#include "HyRect.h"
```

Include dependency graph for ChildRect.h:



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



Classes

- class [ChildRect](#)

A class representing a [HyRect](#) that has a parent [HyRect](#).

9.25.1 Detailed Description

In this file, the header of the [ChildRect](#) class is defined.

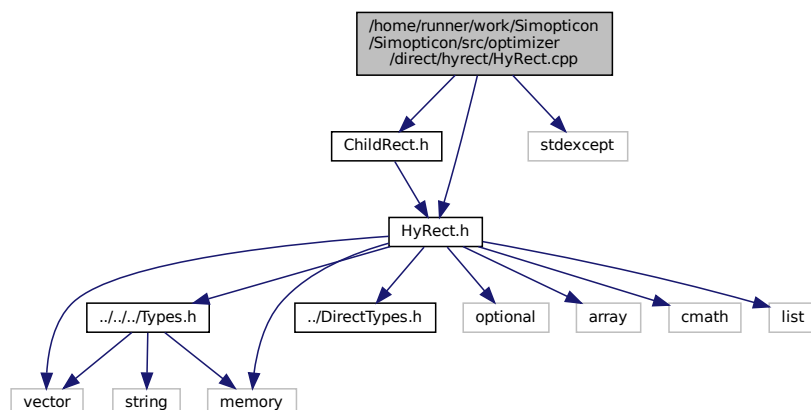
9.26 /home/runner/work/Simopticon/↔

Simopticon/src/optimizer/direct/hyrect/HyRect.cpp File Reference

In this file, the implementation of the [HyRect](#) class is defined.

```
#include "HyRect.h"
#include "ChildRect.h"
#include <stdexcept>
```

Include dependency graph for HyRect.cpp:



9.26.1 Detailed Description

In this file, the implementation of the [HyRect](#) class is defined.

Author

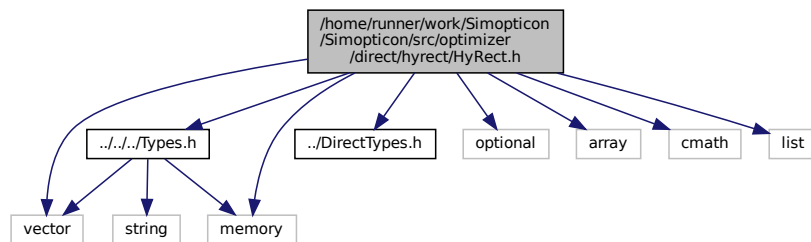
Per Natzschka

9.27 /home/runner/work/Simopticon/↔ Simopticon/src/optimizer/direct/hyrect/HyRect.h File Reference

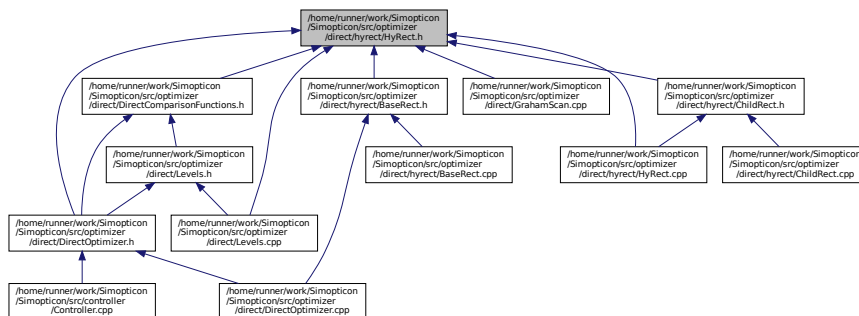
In this file, the header of the [HyRect](#) class is defined.

```
#include "../.../Types.h"
#include "../DirectTypes.h"
#include <optional>
#include <array>
#include <vector>
#include <cmath>
#include <list>
#include <memory>
```

Include dependency graph for HyRect.h:



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



Classes

- class [HyRect](#)

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

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](#).

9.27.1 Detailed Description

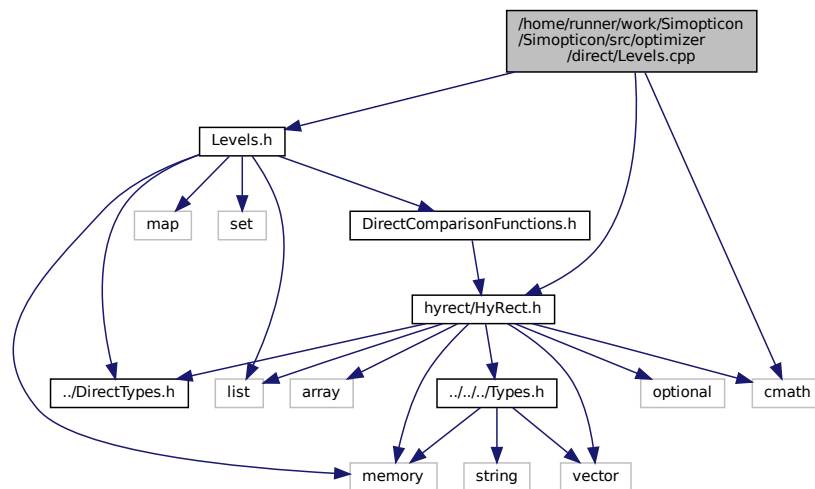
In this file, the header of the [HyRect](#) class is defined.

9.28 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ Levels.cpp File Reference

In this file, the implementation of the [Levels](#) class is defined.

```
#include "Levels.h"
#include "hyrect/HyRect.h"
#include <cmath>
```

Include dependency graph for Levels.cpp:



9.28.1 Detailed Description

In this file, the implementation of the [Levels](#) class is defined.

Author

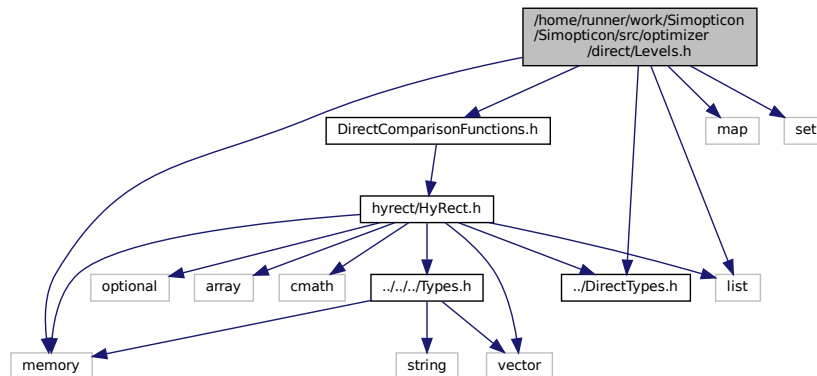
Per Natzschka

9.29 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/↵ Levels.h File Reference

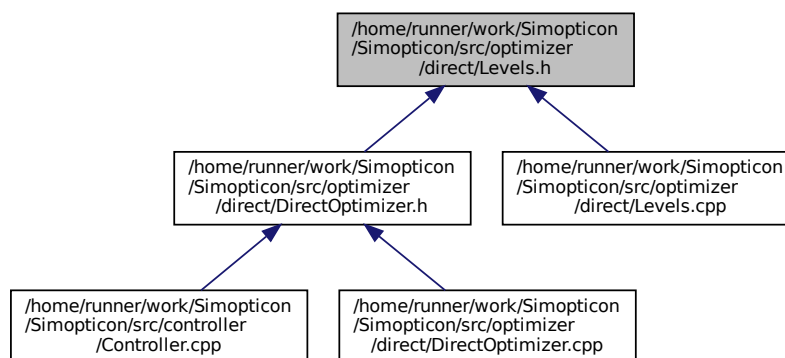
In this file, the header of the [Levels](#) class is defined.

```
#include "DirectComparisonFunctions.h"
#include "DirectTypes.h"
#include <memory>
#include <map>
#include <list>
#include <set>
```

Include dependency graph for Levels.h:



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



Classes

- class [Levels](#)

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

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.

9.29.1 Detailed Description

In this file, the header of the [Levels](#) class is defined.

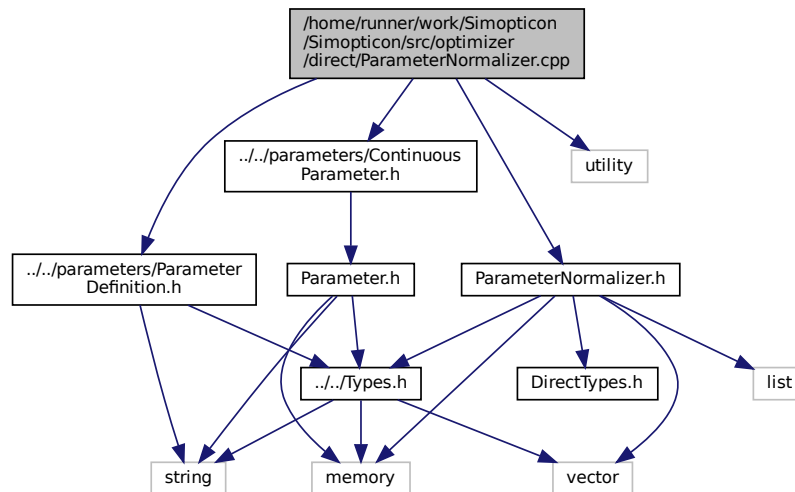
Author

Per Natzschka

9.30 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ParameterNormalizer.cpp File Reference

In this file, the implementation of the [ParameterNormalizer](#) class is defined.

```
#include "ParameterNormalizer.h"
#include <utility>
#include "../..//parameters/ContinuousParameter.h"
#include "../..//parameters/ParameterDefinition.h"
Include dependency graph for ParameterNormalizer.cpp:
```



9.30.1 Detailed Description

In this file, the implementation of the [ParameterNormalizer](#) class is defined.

Author

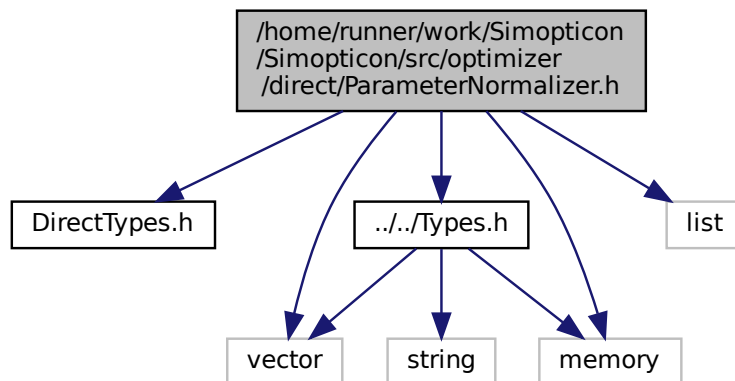
Per Natzschka

9.31 /home/runner/work/Simopticon/Simopticon/src/optimizer/direct/ParameterNormalizer.h File Reference

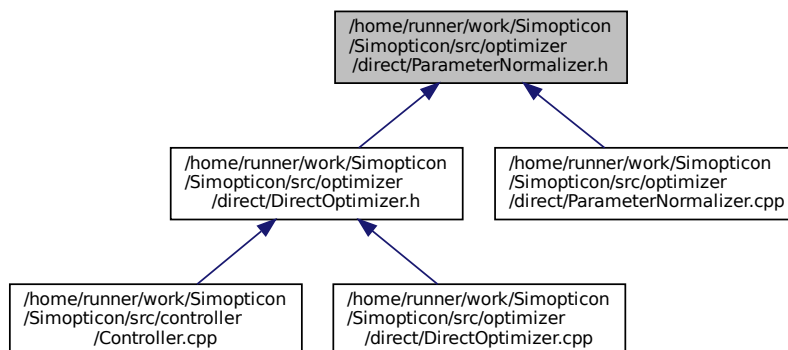
In this file, the header of the [ParameterNormalizer](#) class is defined.

```
#include "DirectTypes.h"
#include "../..//Types.h"
#include <list>
#include <vector>
#include <memory>
```

Include dependency graph for ParameterNormalizer.h:



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



Classes

- class [ParameterNormalizer](#)

A class used for transforming parameters between the actual [Parameter](#) space and the unit hypercube used in DIRECT algorithm.

9.31.1 Detailed Description

In this file, the header of the [ParameterNormalizer](#) class is defined.

Author

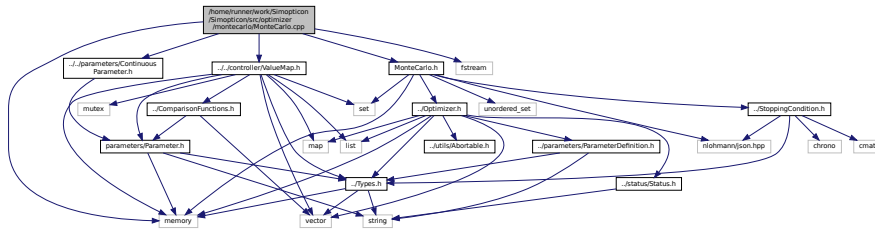
Per Natzschka

9.32 /home/runner/work/Simopticon/↵ Simopticon/src/optimizer/montecarlo/MonteCarlo.cpp File Reference

In this file, the implementation of the [MonteCarlo](#) optimizer class is defined.

```
#include "MonteCarlo.h"
#include "../..//controller/ValueMap.h"
#include "../..//parameters/ContinuousParameter.h"
#include <memory>
#include <fstream>
```

Include dependency graph for MonteCarlo.cpp:



9.32.1 Detailed Description

In this file, the implementation of the [MonteCarlo](#) optimizer class is defined.

Author

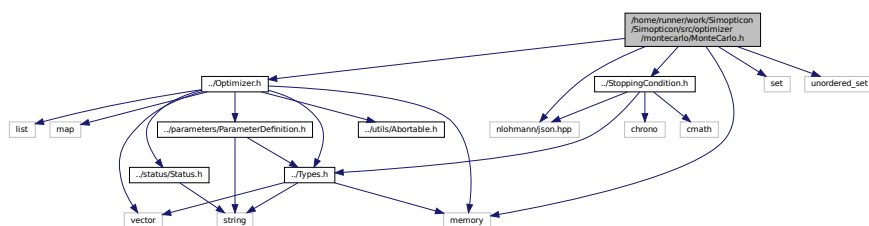
Burkhard Hensel

9.33 /home/runner/work/Simopticon/↵ Simopticon/src/optimizer/montecarlo/MonteCarlo.h File Reference

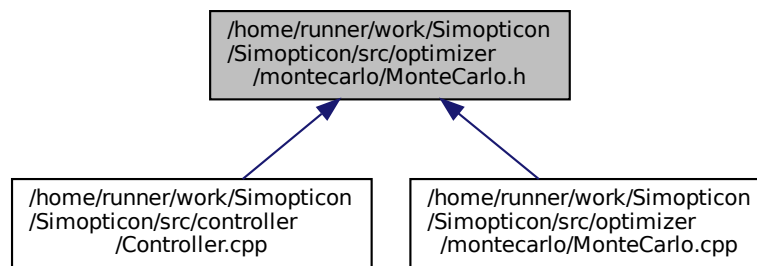
In this file, the header of the [MonteCarlo](#) class is defined.

```
#include "../Optimizer.h"
#include "../StoppingCondition.h"
#include "nlohmann/json.hpp"
#include <set>
#include <unordered_set>
#include <memory>
```

Include dependency graph for MonteCarlo.h:



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



Classes

- class [MonteCarlo](#)

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

9.33.1 Detailed Description

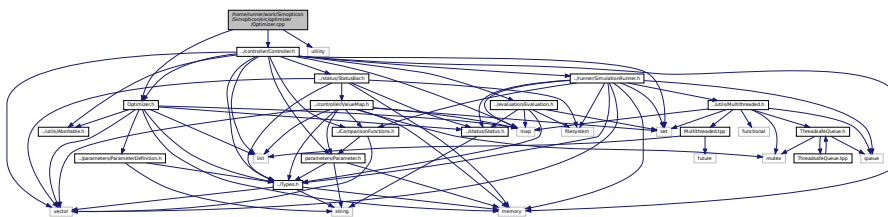
In this file, the header of the [MonteCarlo](#) class is defined.

9.34 /home/runner/work/Simopticon/Simopticon/src/optimizer/↔ Optimizer.cpp File Reference

In this file, the implementation of the [Optimizer](#) class is defined.

```
#include "Optimizer.h"
#include "../controller/Controller.h"
#include <utility>
```

Include dependency graph for Optimizer.cpp:



9.34.1 Detailed Description

In this file, the implementation of the [Optimizer](#) class is defined.

Author

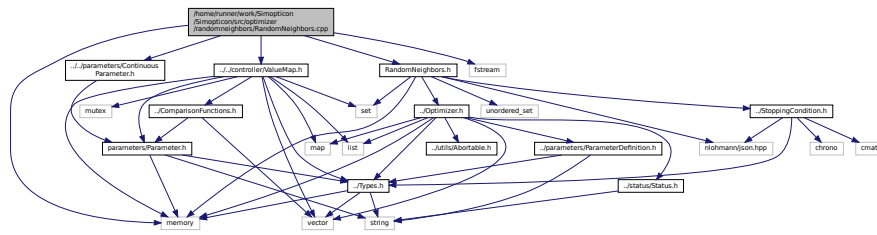
Per Natzschka

9.35 /home/runner/work/Simopticon/Simopticon/src/optimizer/↔ Optimizer.h File Reference

In this file, the header of the [Optimizer](#) class is defined.


```
#include <fstream>
```

Include dependency graph for RandomNeighbors.cpp:



9.36.1 Detailed Description

In this file, the implementation of the [RandomNeighbors](#) optimizer class is defined.

Author

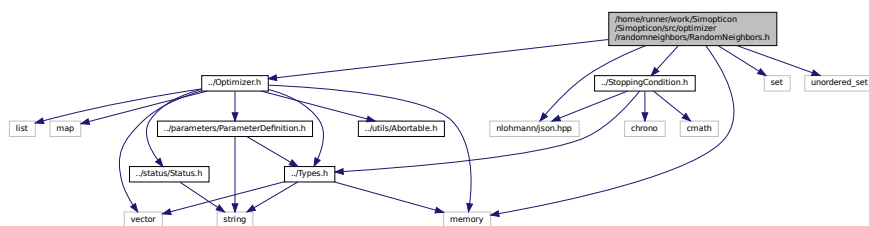
Burkhard Hensel

9.37 /home/runner/work/Simopticon/↵ Simopticon/src/optimizer/randomneighbors/RandomNeighbors.h File Reference

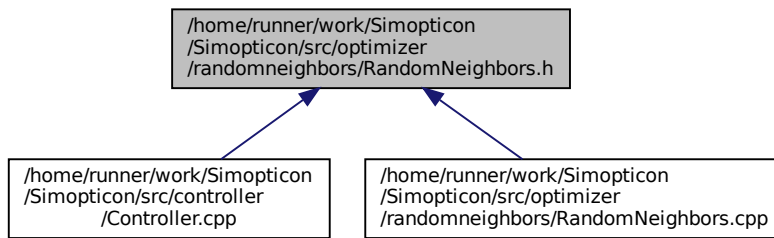
In this file, the header of the [RandomNeighbors](#) optimizer class is defined.

```
#include "../Optimizer.h"
#include "../StoppingCondition.h"
#include "nlohmann/json.hpp"
#include <set>
#include <unordered_set>
#include <memory>
```

Include dependency graph for RandomNeighbors.h:



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



Classes

- class [RandomNeighbors](#)

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

9.37.1 Detailed Description

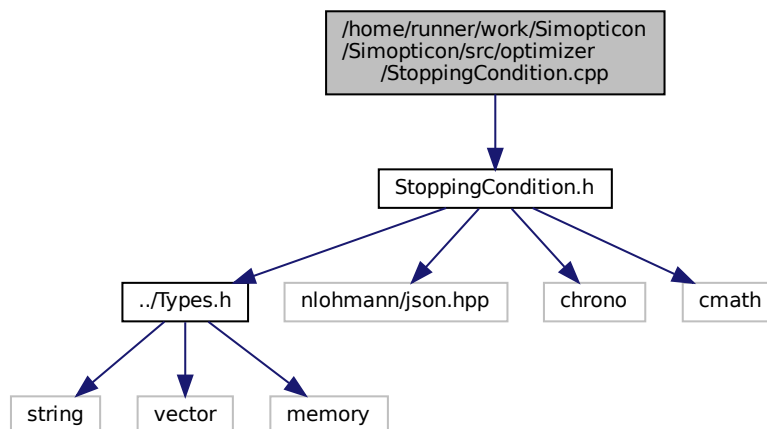
In this file, the header of the [RandomNeighbors](#) optimizer class is defined.

9.38 /home/runner/work/Simopticon/Simopticon/src/optimizer/↔ StoppingCondition.cpp File Reference

In this file, the implementation of the [StoppingCondition](#) class is defined.

```
#include "StoppingCondition.h"
```

Include dependency graph for `StoppingCondition.cpp`:



9.38.1 Detailed Description

In this file, the implementation of the [StoppingCondition](#) class is defined.

Author

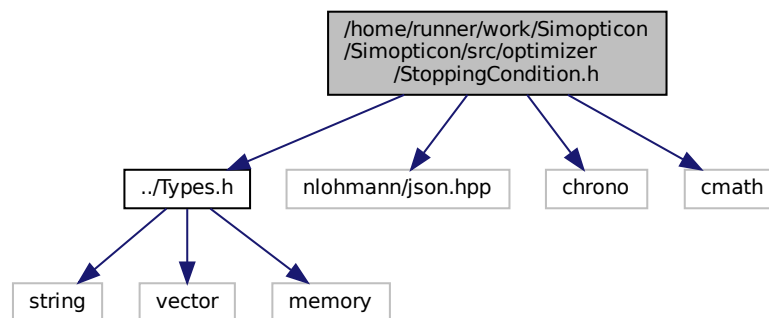
Per Natzschka

9.39 /home/runner/work/Simopticon/Simopticon/src/optimizer/↵ StoppingCondition.h File Reference

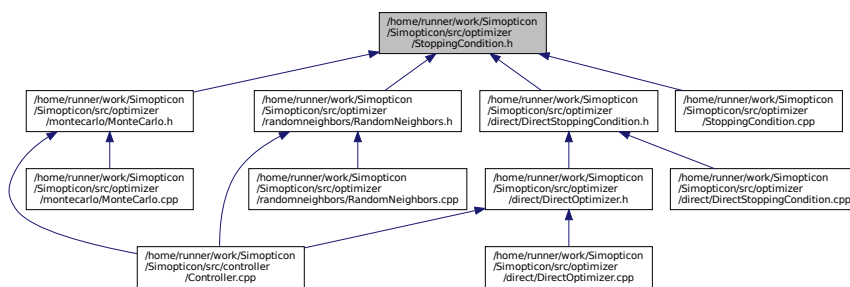
In this file, the header of the [StoppingCondition](#) class is defined.

```
#include "../Types.h"
#include "nlohmann/json.hpp"
#include <chrono>
#include <cmath>
```

Include dependency graph for StoppingCondition.h:



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



Classes

- class [StoppingCondition](#)

A class used for deciding whether the optimization should be stopped.

9.39.1 Detailed Description

In this file, the header of the [StoppingCondition](#) class is defined.

9.40 /home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousParameter.cpp File Reference

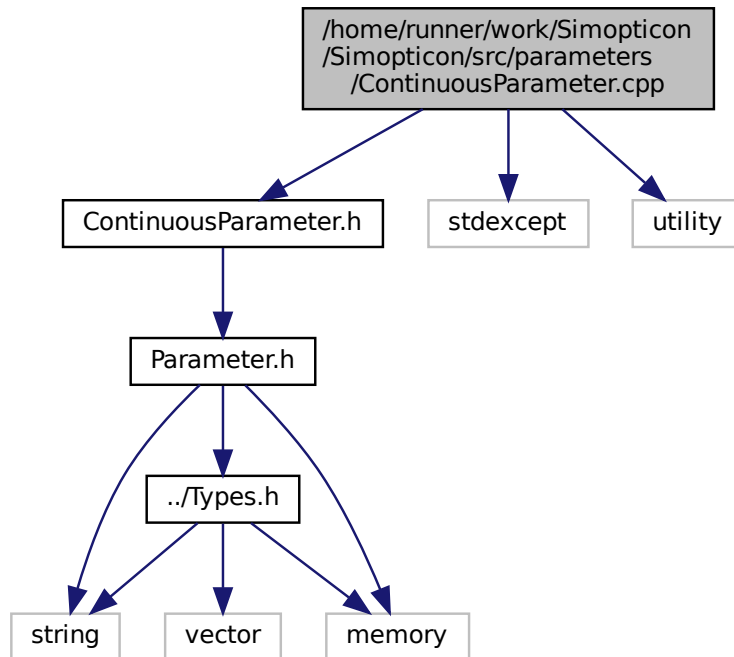
In this file, the implementation of the [ContinuousParameter](#) class is defined.

```
#include "ContinuousParameter.h"
```

```
#include <stdexcept>
```

```
#include <utility>
```

Include dependency graph for ContinuousParameter.cpp:



9.40.1 Detailed Description

In this file, the implementation of the [ContinuousParameter](#) class is defined.

Author

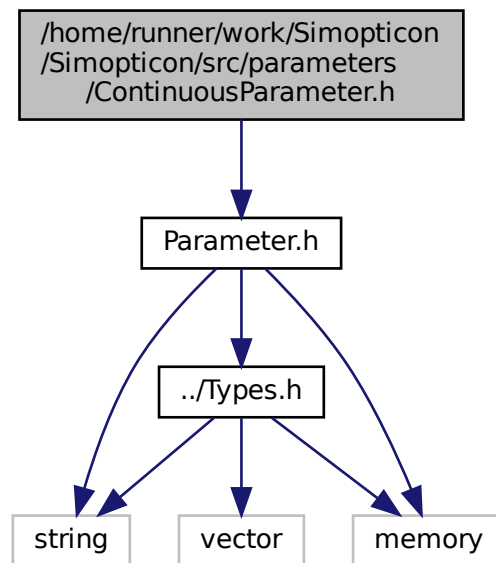
Per Natzschka

9.41 /home/runner/work/Simopticon/Simopticon/src/parameters/↔ ContinuousParameter.h File Reference

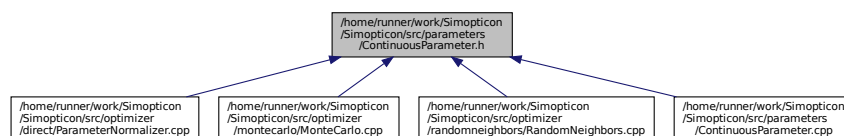
In this file, the header of the [ContinuousParameter](#) class is defined.

```
#include "Parameter.h"
```

Include dependency graph for ContinuousParameter.h:



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



Classes

- class [ContinuousParameter](#)

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

9.41.1 Detailed Description

In this file, the header of the [ContinuousParameter](#) class is defined.

9.42 /home/runner/work/Simopticon/Simopticon/src/parameters/↵ DiscreteParameter.cpp File Reference

In this file, the implementation of the [DiscreteParameter](#) class is defined.

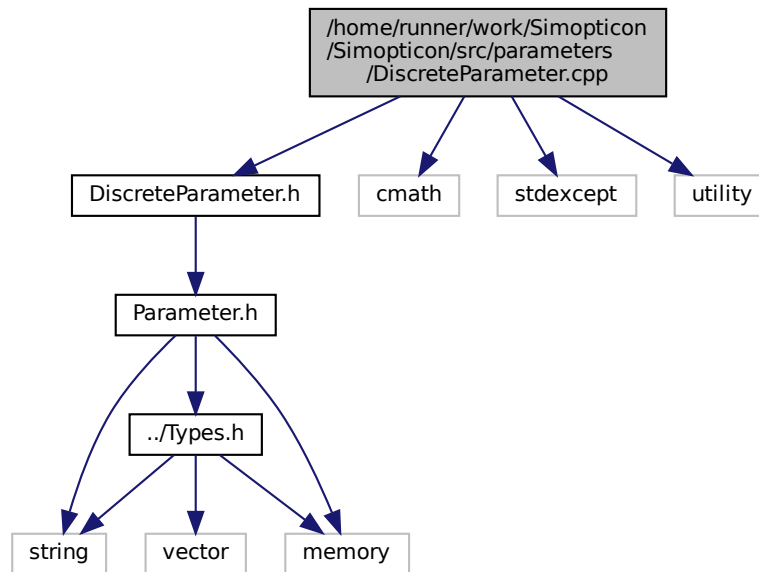
```
#include "DiscreteParameter.h"
```

```
#include <cmath>
```

```
#include <stdexcept>
```

```
#include <utility>
```

Include dependency graph for DiscreteParameter.cpp:



9.42.1 Detailed Description

In this file, the implementation of the [DiscreteParameter](#) class is defined.

Author

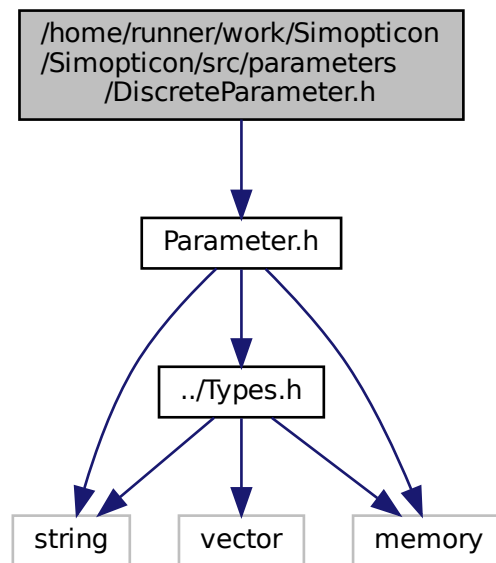
Per Natzschka

9.43 `/home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParameter.h` File Reference

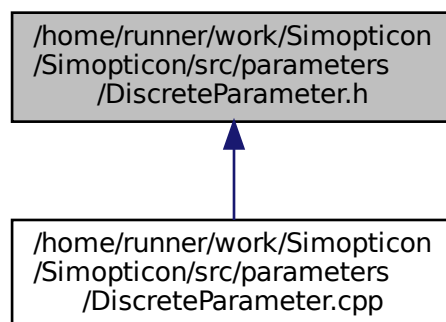
In this file, the header of the `DiscreteParameter` class is defined.

```
#include "Parameter.h"
```

Include dependency graph for `DiscreteParameter.h`:



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



Classes

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

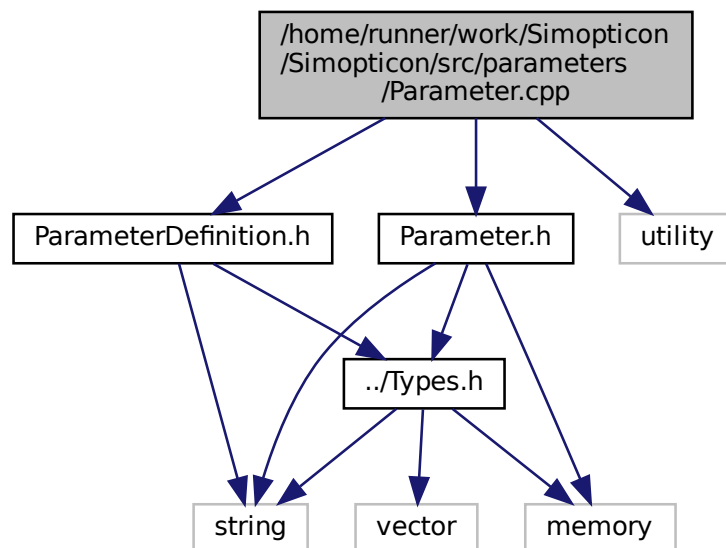
9.43.1 Detailed Description

In this file, the header of the [DiscreteParameter](#) class is defined.

9.44 /home/runner/work/Simopticon/Simopticon/src/parameters/Parameter.cpp File Reference

In this file, the implementation of the [Parameter](#) class is defined.

```
#include "Parameter.h"
#include "ParameterDefinition.h"
#include <utility>
Include dependency graph for Parameter.cpp:
```



9.44.1 Detailed Description

In this file, the implementation of the [Parameter](#) class is defined.

Author

Per Natzschka

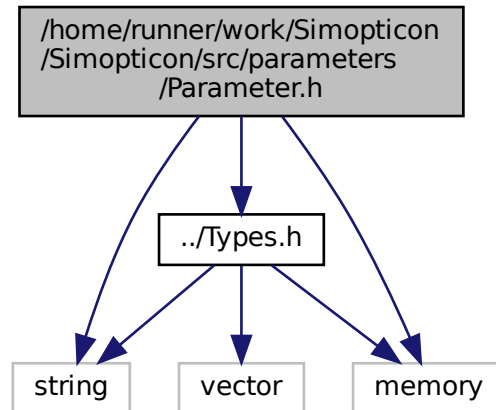
9.45 /home/runner/work/Simopticon/Simopticon/src/parameters/Parameter.h File Reference

In this file, the header of the [Parameter](#) class is defined.

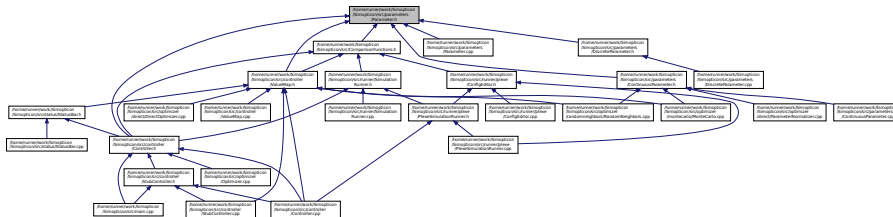
```
#include "../Types.h"
#include <string>
```

```
#include <memory>
```

Include dependency graph for Parameter.h:



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



Classes

- class [Parameter](#)

A class acting as the container of the value of a parameter defined by a [ParameterDefinition](#).

9.45.1 Detailed Description

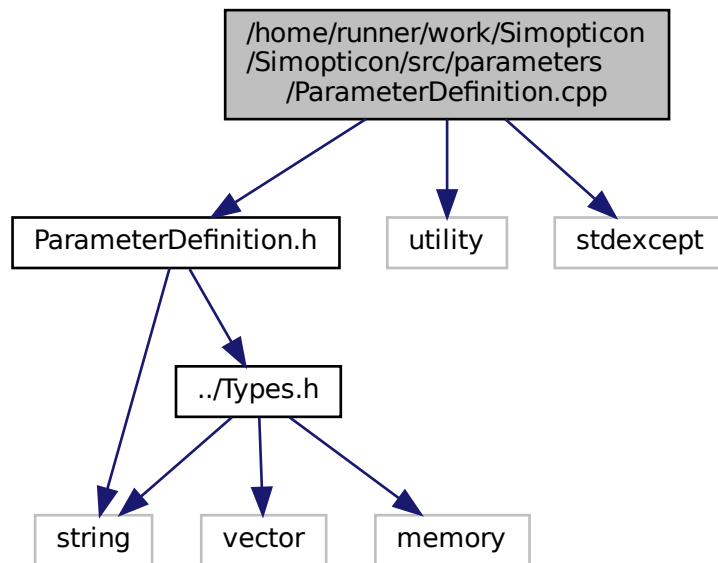
In this file, the header of the [Parameter](#) class is defined.

9.46 `/home/runner/work/Simopticon/Simopticon/src/parameters/ParameterDefinition.cpp` File Reference

In this file, the implementation of the [ParameterDefinition](#) class is defined.

```
#include "ParameterDefinition.h"
#include <utility>
#include <stdexcept>
```

Include dependency graph for ParameterDefinition.cpp:



9.46.1 Detailed Description

In this file, the implementation of the `ParameterDefinition` class is defined.

Author

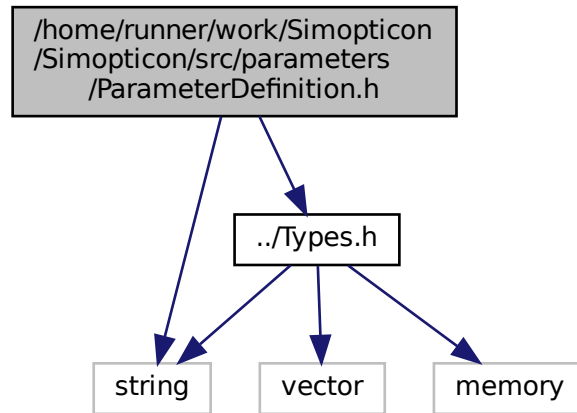
Per Natzschka

9.47 /home/runner/work/Simopticon/Simopticon/src/parameters/ParameterDefinition.h File Reference

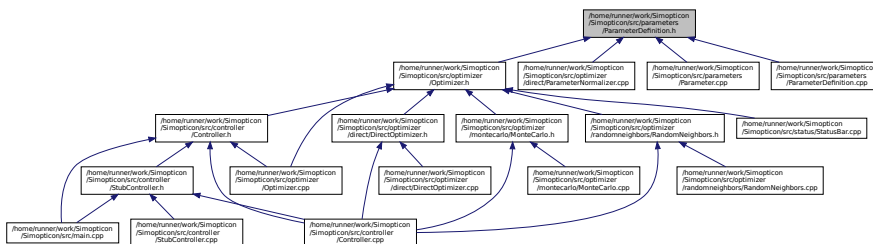
In this file, the header of the `ParameterDefinition` class is defined.

```
#include "../Types.h"  
#include <string>
```

Include dependency graph for `ParameterDefinition.h`:



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



Classes

- class [ParameterDefinition](#)

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

9.47.1 Detailed Description

In this file, the header of the [ParameterDefinition](#) class is defined.

9.48 /home/runner/work/Simopticon/Simopticon/src/runner/plexer/↵ ConfigEditor.cpp File Reference

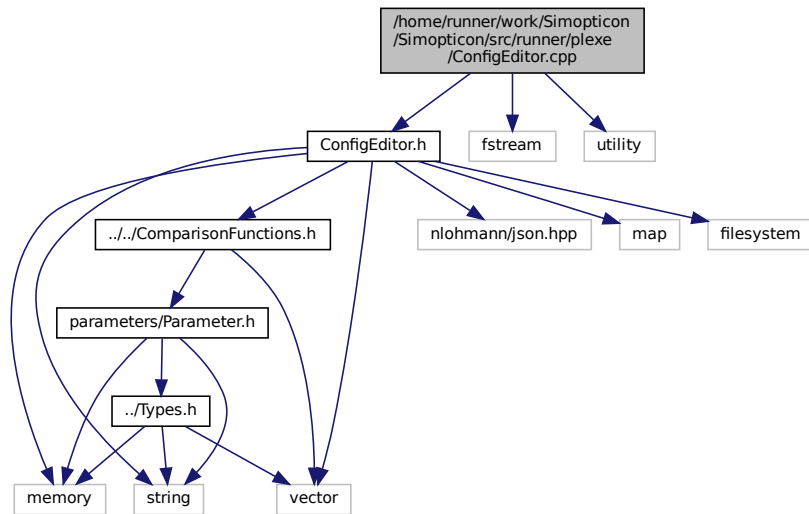
In this file, the implementation of the [ConfigEditor](#) class is defined.

```

#include "ConfigEditor.h"
#include <fstream>
#include <utility>

```

Include dependency graph for ConfigEditor.cpp:



9.48.1 Detailed Description

In this file, the implementation of the [ConfigEditor](#) class is defined.

Author

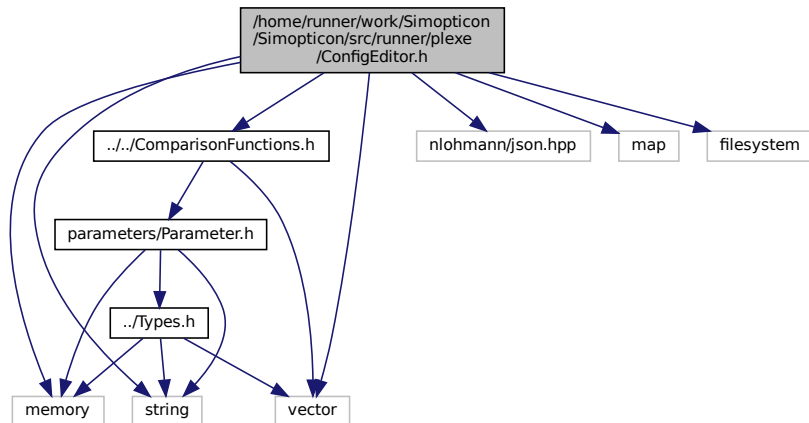
Per Natzschka

9.49 /home/runner/work/Simopticon/Simopticon/src/runner/plex/↵ ConfigEditor.h File Reference

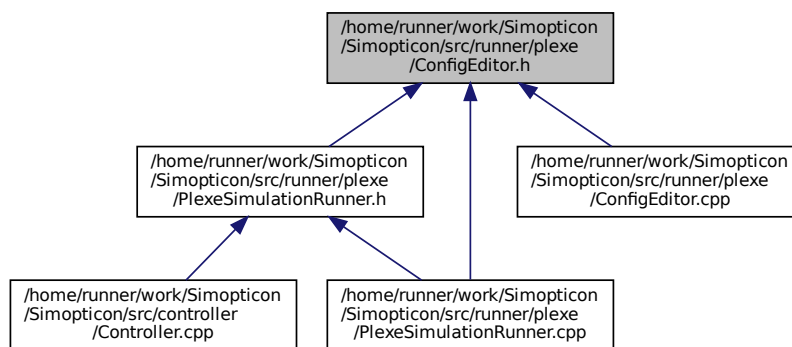
In this file, the header of the [ConfigEditor](#) class is defined.

```
#include "../../ComparisonFunctions.h"  
#include "nlohmann/json.hpp"  
#include <memory>  
#include <string>  
#include <vector>  
#include <map>  
#include <filesystem>
```

Include dependency graph for ConfigEditor.h:



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



Classes

- class [ConfigEditor](#)

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

9.49.1 Detailed Description

In this file, the header of the [ConfigEditor](#) class is defined.

9.50 /home/runner/work/Simopticon/Simopticon/src/runner/plex/← PlexeSimulationRunner.cpp File Reference

In this file, the implementation of the [PlexeSimulationRunner](#) class is defined.

```

#include "PlexeSimulationRunner.h"
#include "ConfigEditor.h"
#include "../../utils/CommandLine.h"
#include <utility>

```


Classes

- class [PlexeSimulationRunner](#)

A class capable of starting platooning simulations in the *Plexe* framework with given parameterCombinations.

9.51.1 Detailed Description

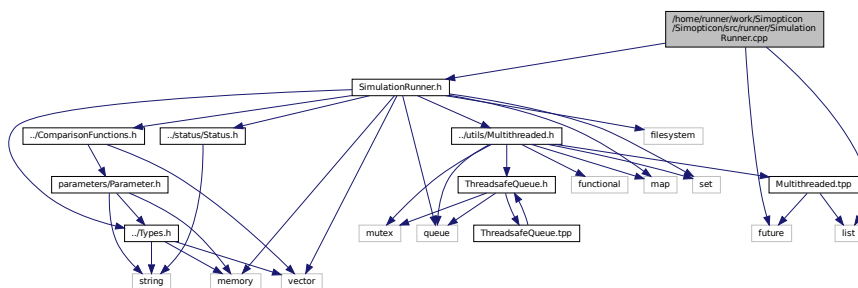
In this file, the header of the [PlexeSimulationRunner](#) class is defined.

9.52 /home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.cpp File Reference

In this file, the implementation of the [SimulationRunner](#) class is defined.

```
#include "SimulationRunner.h"
#include <future>
#include <list>
```

Include dependency graph for SimulationRunner.cpp:



9.52.1 Detailed Description

In this file, the implementation of the [SimulationRunner](#) class is defined.

Author

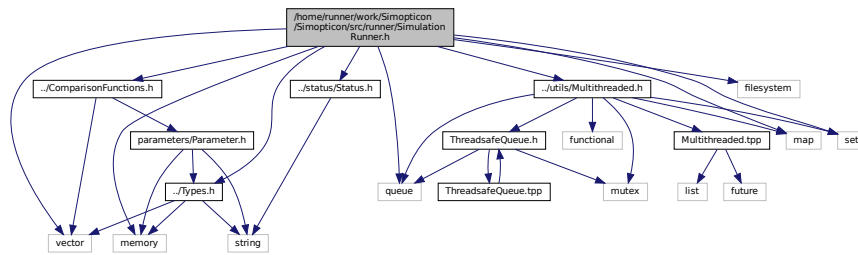
Per Natzschka

9.53 /home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.h File Reference

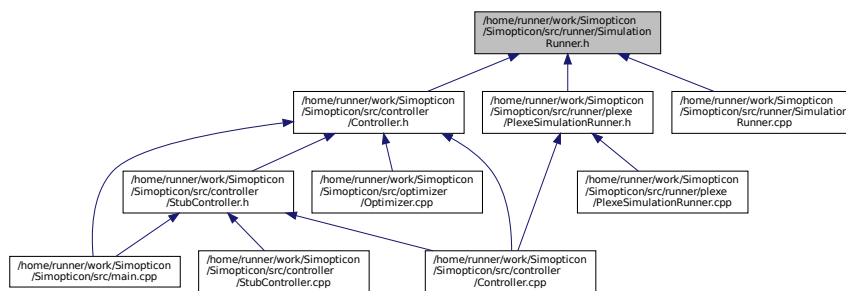
In this file, the header of the [SimulationRunner](#) class is defined.

```
#include "../Types.h"
#include "../ComparisonFunctions.h"
#include "../utils/Multithreaded.h"
#include "../status/Status.h"
#include <vector>
#include <set>
#include <map>
#include <memory>
#include <filesystem>
#include <queue>
```


Include dependency graph for SimulationRunner.h:



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



Classes

- class [SimulationRunner](#)

A class capable of running simulations with certain parameterCombinations.

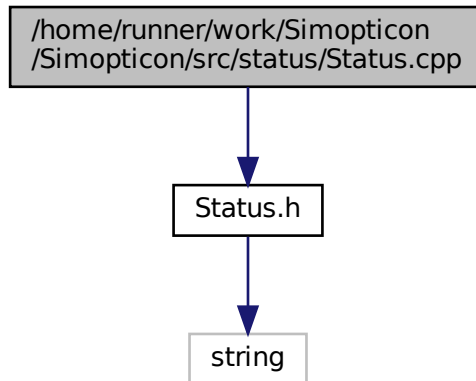
9.53.1 Detailed Description

In this file, the header of the [SimulationRunner](#) class is defined.

9.54 /home/runner/work/Simopticon/Simopticon/src/status/Status.cpp File Reference

In this file, the implementation of the [Status](#) class is defined.

```
#include "Status.h"  
Include dependency graph for Status.cpp:
```



9.54.1 Detailed Description

In this file, the implementation of the [Status](#) class is defined.

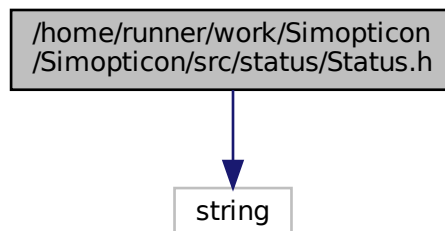
Author

Per Natzschka

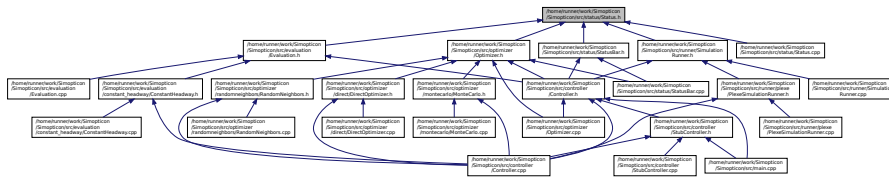
9.55 `/home/runner/work/Simopticon/Simopticon/src/status/Status.h` File Reference

In this file, the header of the [Status](#) class is defined.

```
#include <string>  
Include dependency graph for Status.h:
```



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



Classes

- class [Status](#)

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

9.55.1 Detailed Description

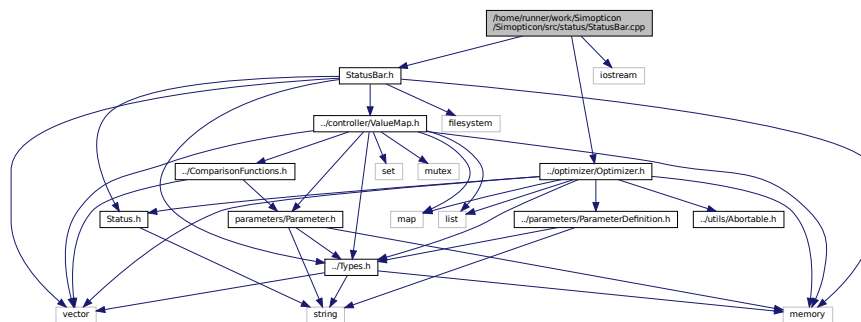
In this file, the header of the [Status](#) class is defined.

9.56 /home/runner/work/Simopticon/Simopticon/src/status/StatusBar.cpp File Reference

In this file, the implementation of the [StatusBar](#) class is defined.

```
#include "StatusBar.h"
#include "../optimizer/Optimizer.h"
#include <iostream>
```

Include dependency graph for StatusBar.cpp:



9.56.1 Detailed Description

In this file, the implementation of the [StatusBar](#) class is defined.

Author

Per Natzschka

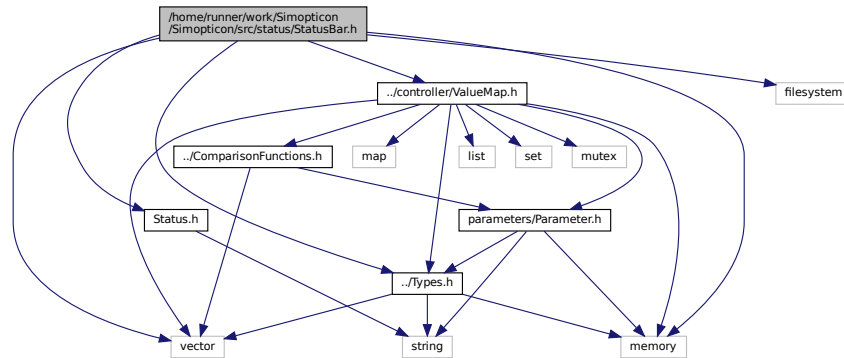
9.57 /home/runner/work/Simopticon/Simopticon/src/status/StatusBar.h File Reference

In this file, the header of the [StatusBar](#) class is defined.

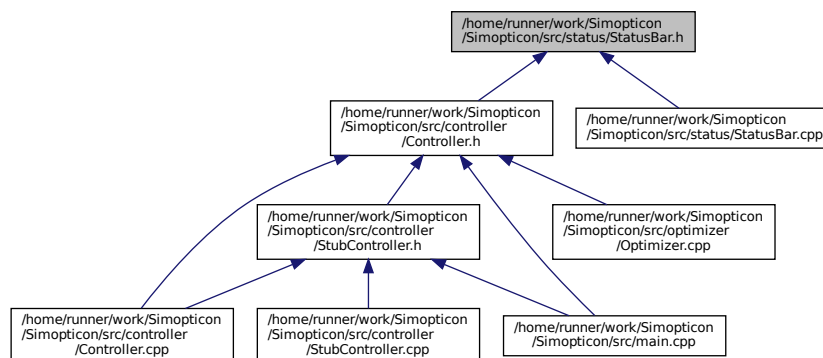
```
#include "../Types.h"
#include "../controller/ValueMap.h"
```

```
#include "Status.h"
#include <memory>
#include <vector>
#include <filesystem>
```

Include dependency graph for StatusBar.h:



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



Classes

- 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.

9.57.1 Detailed Description

In this file, the header of the `StatusBar` class is defined.

Author

Per Natzschka

9.58 /home/runner/work/Simopticon/Simopticon/src/Types.h File Reference

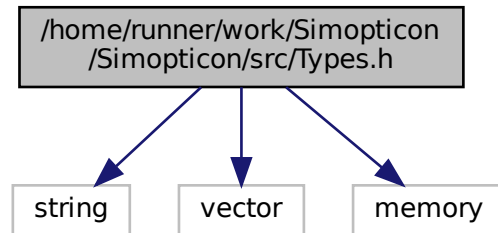
In this file, types are defined which should be used across the whole framework.

```
#include <string>
```

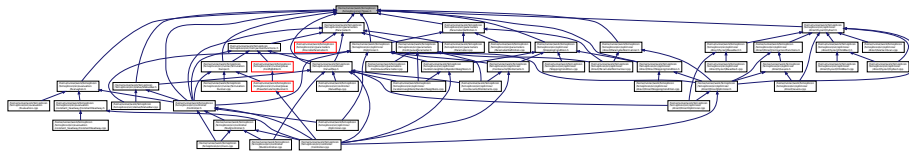
```
#include <vector>
```

```
#include <memory>
```

Include dependency graph for Types.h:



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



Typedefs

- typedef `std::vector< std::shared_ptr< Parameter > >` `parameterCombination`
A complex type representing a vector in parameter space.
- 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.

9.58.1 Detailed Description

In this file, types are defined which should be used across the whole framework.

Author

Per Natzschka

9.58.2 Typedef Documentation

9.58.2.1 coordinate

```
typedef double coordinate
```

A floating point type used to represent [Parameter](#) values.

Definition at line 32 of file Types.h.

9.58.2.2 functionValue

```
typedef long double functionValue
```

A floating point type containing the value of an optimized function.

Definition at line 27 of file Types.h.

9.58.2.3 parameterCombination

```
typedef std::vector<std::shared_ptr<Parameter> > parameterCombination
```

A complex type representing a vector in parameter space.

Definition at line 22 of file Types.h.

9.58.2.4 runId

```
typedef std::string runId
```

An identifier that makes different simulation runs in one result file folder distinguishable.

Uniqueness is not being asserted.

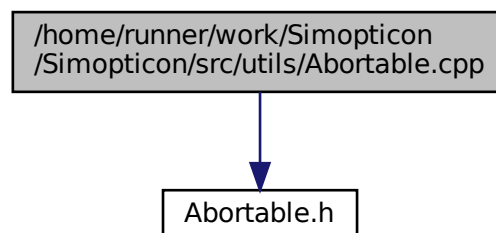
Definition at line 38 of file Types.h.

9.59 /home/runner/work/Simopticon/Simopticon/src/utils/Abortable.cpp File Reference

In this file, the implementation of the [Abortable](#) class is defined.

```
#include "Abortable.h"
```

Include dependency graph for Abortable.cpp:



9.59.1 Detailed Description

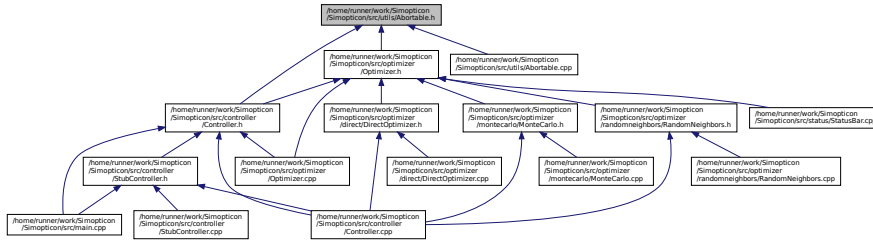
In this file, the implementation of the [Abortable](#) class is defined.

Author

Per Natzschka

9.60 /home/runner/work/Simopticon/Simopticon/src/utills/Abortable.h File Reference

In this file, the header of the `Abortable` class is defined.
 This graph shows which files directly or indirectly include this file:



Classes

- class `Abortable`
A simple interface for classes that encapsulate abortable processes.

9.60.1 Detailed Description

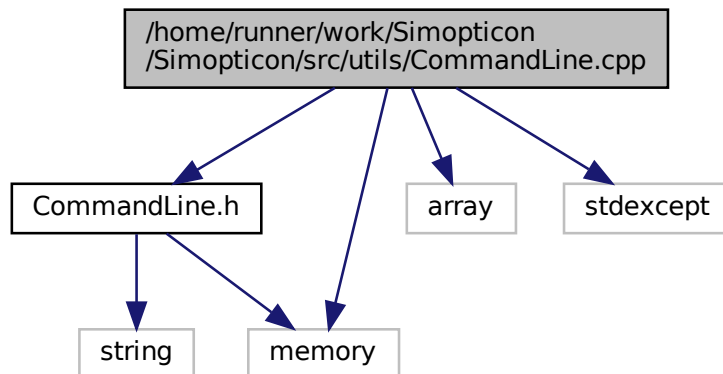
In this file, the header of the `Abortable` class is defined.

9.61 /home/runner/work/Simopticon/Simopticon/src/utills/CommandLine.cpp File Reference

In this file, the implementation of the `CommandLine` class is defined.

```
#include "CommandLine.h"
#include <array>
#include <memory>
#include <stdexcept>
```

Include dependency graph for `CommandLine.cpp`:



9.61.1 Detailed Description

In this file, the implementation of the [CommandLine](#) class is defined.

Author

Per Natzschka

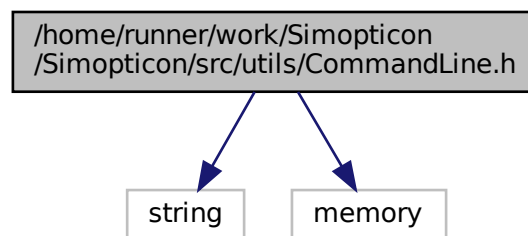
9.62 /home/runner/work/Simopticon/Simopticon/src/utils/CommandLine.h File Reference

In this file, the header of the [CommandLine](#) class is defined.

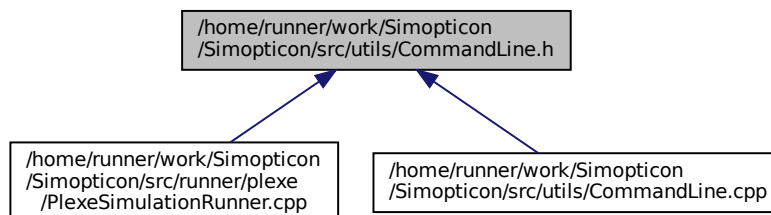
```
#include <string>
```

```
#include <memory>
```

Include dependency graph for CommandLine.h:



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



Classes

- class [CommandLine](#)

A class containing functionality for executing commands on UNIX shell.

9.62.1 Detailed Description

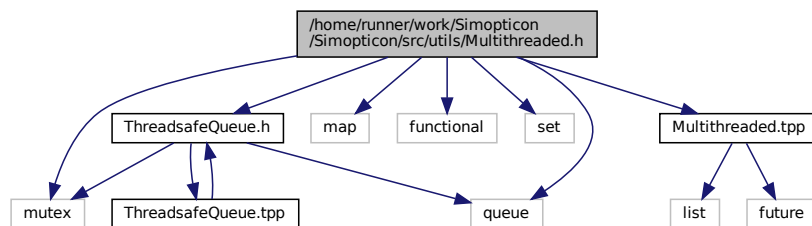
In this file, the header of the [CommandLine](#) class is defined.

9.63 /home/runner/work/Simopticon/Simopticon/src/Utils/Multithreaded.h File Reference

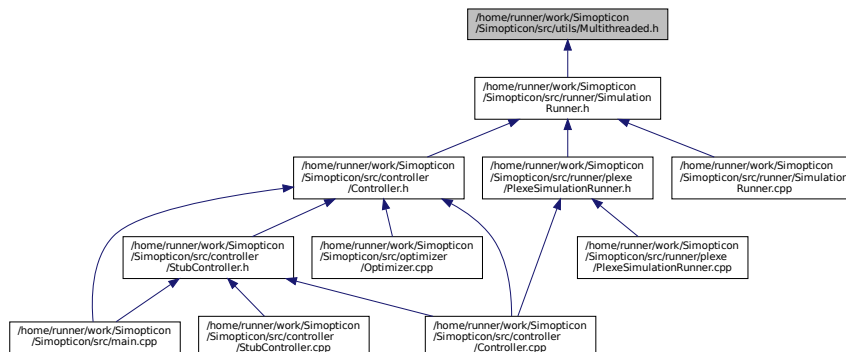
In this file, the header of the `Multithreaded` class is defined.

```
#include <mutex>
#include <queue>
#include <map>
#include <functional>
#include <set>
#include "ThreadsafeQueue.h"
#include "Multithreaded.hpp"
```

Include dependency graph for Multithreaded.h:



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



Classes

- class `Multithreaded< Key, T, Compare, Allocator >`

A class implementing concurrent execution of the same function for different arguments.

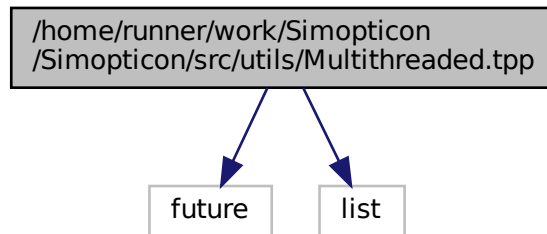
9.63.1 Detailed Description

In this file, the header of the `Multithreaded` class is defined.

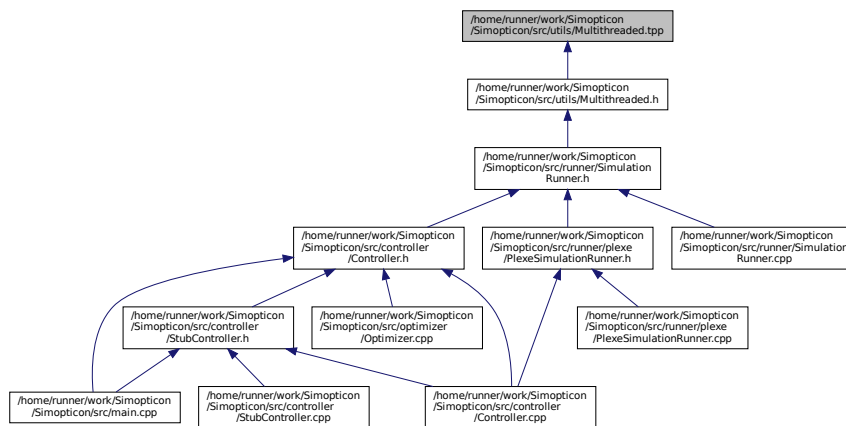
9.64 /home/runner/work/Simopticon/Simopticon/src/Utils/Multithreaded.hpp File Reference

In this file, the implementation of the `Multithreaded` class is defined.

```
#include <future>
#include <list>
Include dependency graph for Multithreaded.hpp:
```



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



9.64.1 Detailed Description

In this file, the implementation of the [Multithreaded](#) class is defined.

Author

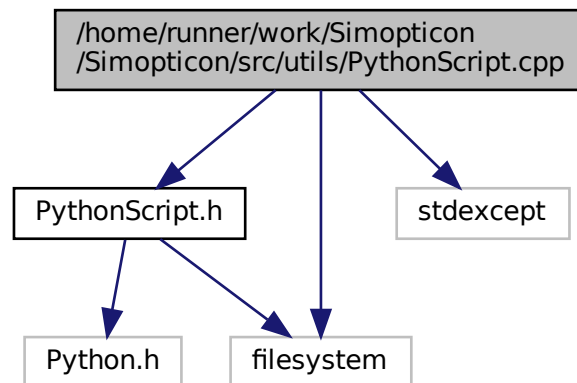
Per Natzschka

9.65 /home/runner/work/Simopticon/Simopticon/src/utis/PythonScript.cpp File Reference

In this file, the implementation of the [PythonScript](#) class is defined.

```
#include "PythonScript.h"
#include <stdexcept>
#include <filesystem>
```

Include dependency graph for PythonScript.cpp:



9.65.1 Detailed Description

In this file, the implementation of the [PythonScript](#) class is defined.

Author

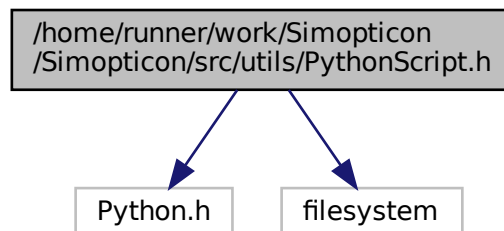
Per Natzschka

9.66 /home/runner/work/Simopticon/Simopticon/src/Utils/PythonScript.h File Reference

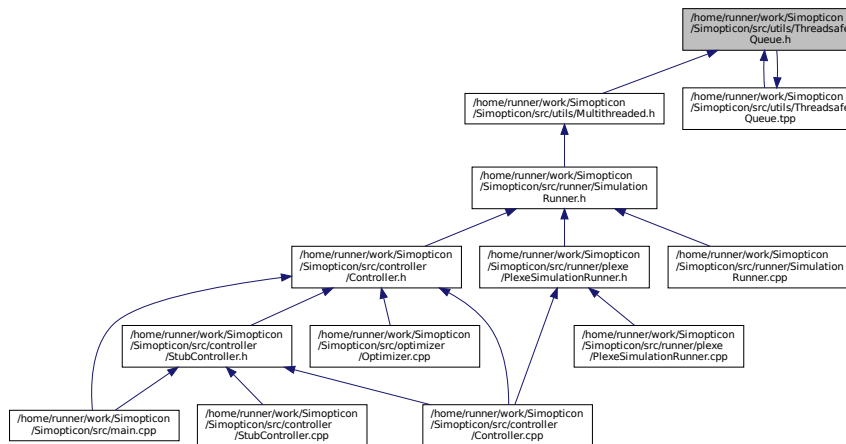
In this file, the header of the [PythonScript](#) class is defined.

```
#include <Python.h>  
#include <filesystem>
```

Include dependency graph for PythonScript.h:



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



Classes

- class [ThreadsafeQueue< Key >](#)

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

9.67.1 Detailed Description

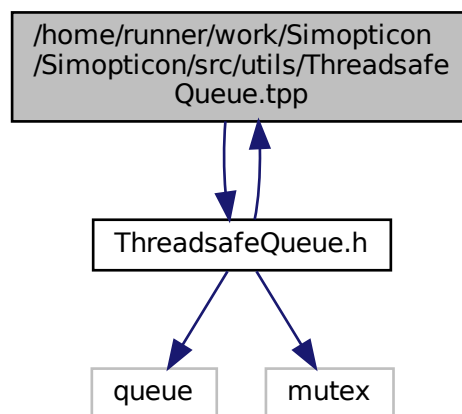
In this file, the header of the ThreadSafeQueue class is defined.

9.68 /home/runner/work/Simopticon/Simopticon/src/utlis/ThreadsafeQueue.tpp File Reference

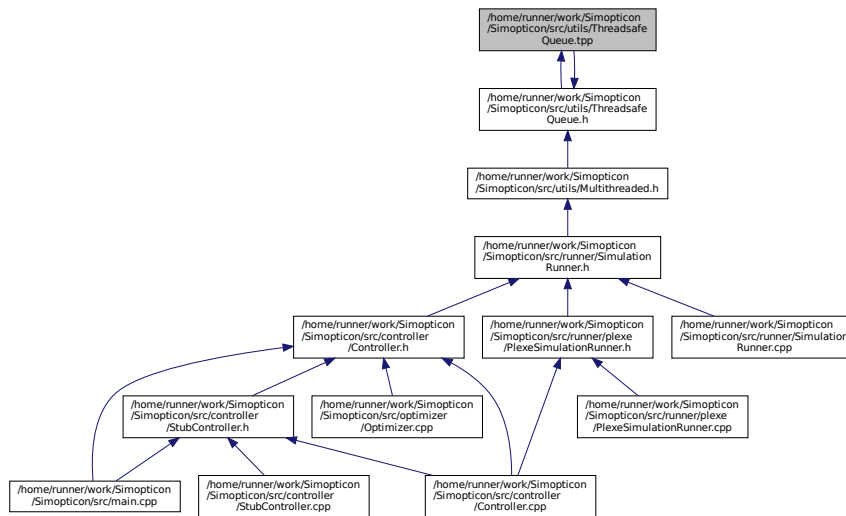
In this file, the implementation of the ThreadSafeQueue class is defined.

```
#include "ThreadsafeQueue.h"
```

Include dependency graph for ThreadsafeQueue.tpp:



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



9.68.1 Detailed Description

In this file, the implementation of the ThreadSafeQueue class is defined.

Author

Per Natzschka

Index

[/home/runner/work/Simopticon/Simopticon/src/ComparisonFunctions](#) 169
[149](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/GrahamScan.cpp](#)

[/home/runner/work/Simopticon/Simopticon/src/Types.h](#), [169](#)
[201](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Levels.cpp](#),

[/home/runner/work/Simopticon/Simopticon/src/controller/Controller.cpp](#) 175
[150](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Levels.h](#),

[/home/runner/work/Simopticon/Simopticon/src/controller/Controller.h](#) 175
[151](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Parameter](#)

[/home/runner/work/Simopticon/Simopticon/src/controller/StubController.cpp](#),
[152](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/Parameter](#)

[/home/runner/work/Simopticon/Simopticon/src/controller/StubController.h](#),
[154](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/Base](#)

[/home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.cpp](#) 170
[155](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/Base](#)

[/home/runner/work/Simopticon/Simopticon/src/controller/ValueMap.h](#) 171
[155](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/Child](#)

[/home/runner/work/Simopticon/Simopticon/src/evaluation/Evaluation.cpp](#),
[159](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/Child](#)

[/home/runner/work/Simopticon/Simopticon/src/evaluation/Evaluation.h](#) 172
[160](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/HyR](#)

[/home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.cpp](#),
[158](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/hyrect/HyR](#)

[/home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/ConstantHeadway.h](#),
[158](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/Monte](#)

[/home/runner/work/Simopticon/Simopticon/src/evaluation/constant_headway/constant_headway.py](#),
[156](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/montecarlo/Monte](#)

[/home/runner/work/Simopticon/Simopticon/src/main.cpp](#), [179](#)
[161](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.cpp](#) 181
[180](#) [/home/runner/work/Simopticon/Simopticon/src/optimizer/randomneighbors](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/Optimizer.h](#) 182
[180](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousPar](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.cpp](#),
[183](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/ContinuousPar](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/StoppingCondition.h](#),
[184](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParam](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectComparisonFunctions.h](#),
[163](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/DiscreteParam](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectOptimizer.cpp](#),
[164](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/Parameter.cpp](#),

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectOptimizer.h](#),
[164](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/Parameter.h](#),

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectStoppingCondition.cpp](#),
[165](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/ParameterDefin](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectStoppingCondition.h](#),
[166](#) [/home/runner/work/Simopticon/Simopticon/src/parameters/ParameterDefin](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/DirectTypes.h](#),
[167](#) [/home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.c](#)

[/home/runner/work/Simopticon/Simopticon/src/optimizer/direct/GrahamScan.cpp](#),
[166](#)

- /home/runner/work/Simopticon/Simopticon/src/runner/SimulationRunner.cpp, 196
- /home/runner/work/Simopticon/Simopticon/src/runner/plex/Controller.cpp, 192
- /home/runner/work/Simopticon/Simopticon/src/runner/plex/Controller.h, 193
- /home/runner/work/Simopticon/Simopticon/src/runner/plex/plexSimulationRunner.cpp, 194
- /home/runner/work/Simopticon/Simopticon/src/runner/plex/plexSimulationRunner.h, 195
- /home/runner/work/Simopticon/Simopticon/src/status/Status.cpp, 197
- /home/runner/work/Simopticon/Simopticon/src/status/Status.h, 198
- /home/runner/work/Simopticon/Simopticon/src/status/StatusBar.cpp, 199
- /home/runner/work/Simopticon/Simopticon/src/status/StatusBar.h, 199
- /home/runner/work/Simopticon/Simopticon/src/Utils/Abortable.cpp, 202
- /home/runner/work/Simopticon/Simopticon/src/Utils/Abortable.h, 203
- /home/runner/work/Simopticon/Simopticon/src/Utils/CommandLine.cpp, 203
- /home/runner/work/Simopticon/Simopticon/src/Utils/CommandLine.h, 204
- /home/runner/work/Simopticon/Simopticon/src/Utils/Multithreaded.h, 205
- /home/runner/work/Simopticon/Simopticon/src/Utils/Multithreaded.cpp, 205
- /home/runner/work/Simopticon/Simopticon/src/Utils/PythonScript.cpp, 206
- /home/runner/work/Simopticon/Simopticon/src/Utils/PythonScript.h, 207
- /home/runner/work/Simopticon/Simopticon/src/Utils/ThreadsafeQueue.h, 208
- /home/runner/work/Simopticon/Simopticon/src/Utils/ThreadsafeQueue.cpp, 209
- ~PythonScript
 - PythonScript, 114
- abort
 - Abortable, 30
 - Controller, 53
- Abortable, 29
 - abort, 30
 - aborted, 30
- aborted
 - Abortable, 30
- ACCURACY
 - StoppingCondition, 134
- activeRects
 - DirectOptimizer, 62
- addActiveRects
 - DirectOptimizer, 59
- addValue
 - ValueMap, 144
- avgValue
 - HyRect, 80
- BaseRect, 60
 - BaseRect, 31
- getSamplingVertices, 31
 - bestVal
- StoppingCondition, 135
 - ChildRect, 32
- plexSimulationRunner.cpp, 194
 - ChildRect, 33
- plexSimulationRunner.h, 195
 - getSamplingVertices, 33
 - operator==, 34
 - parent, 34
- Status.cpp, 197
 - CmpPairVectorSharedParameterFunctionvalue, 34
 - operator(), 35
- Status.h, 198
 - CmpPtrFunctionvalue, 35
 - operator(), 36
- StatusBar.cpp, 199
 - CmpSharedHyrect, 36
 - operator(), 36
- StatusBar.h, 199
 - CmpVectorSharedParameter, 37
 - operator(), 37
- Abortable.cpp, 202
 - CommandLine, 38
- Abortable.h, 203
 - exec, 38
 - CONFIG
- CommandLine.cpp, 203
 - ConfigEditor, 43
 - config
- CommandLine.h, 204
 - ParameterDefinition, 105
 - ConfigEditor, 39
- Multithreaded.h, 205
 - CONFIG, 43
 - ConfigEditor, 40
- Multithreaded.cpp, 205
 - CONTROLLER, 43
 - createConfig, 40
 - deleteConfig, 41
 - DIR, 43
- PythonScript.cpp, 206
 - getConfigPath, 41
 - getControllerOption, 41
 - getDir, 42
- PythonScript.h, 207
 - getResultPath, 42
 - replaceOption, 42
 - RESULTS, 43
 - setResultFiles, 43
- ThreadsafeQueue.h, 208
 - constant_headway, 22
 - constant_headway.py
 - get_constant_headway, 157
 - multithreaded, 157
- ThreadsafeQueue.cpp, 209
 - ConstantHeadway, 44
 - ConstantHeadway, 45
 - getName, 45
 - getStatus, 46
 - getStatusBar, 46
 - NR_THREADS, 47
 - processOutput, 46, 47
 - secureValue, 47
 - usedThreads, 47
- PythonScript
 - ContinuousParameter, 48
 - ContinuousParameter, 49
 - getVal, 49
 - setVal, 50
 - val, 50
- CONTROLLER
 - ConfigEditor, 43

- Controller, 50
 - abort, 53
 - Controller, 53
 - evaluate, 53
 - evaluation, 55
 - getValueMap, 53
 - keepFiles, 55
 - optimizer, 55
 - printValues, 56
 - removeOldResultfiles, 54
 - requestValues, 54
 - run, 54
 - runner, 56
 - runSimulations, 54
 - saveValues, 55
 - statusBar, 56
 - statusInterval, 56
 - topResults, 56
 - updateStatus, 55
 - valueMap, 56
- controller, 19
 - Optimizer, 97
 - stepState, 19
- Controller.cpp
 - getConfigByPath, 151
- Controller::stepstate, 129
 - currentStep, 130
 - get, 130
 - next, 130
 - stepChanged, 130
- coordinate
 - Types.h, 201
- createConfig
 - ConfigEditor, 40
- ctr
 - main.cpp, 162
- currentLevel
 - Levels, 83
- currentStep
 - Controller::stepstate, 130
- D
 - DirectOptimizer, 62
 - HyRect, 80
- definition
 - Parameter, 102
- deleteConfig
 - ConfigEditor, 41
- denormalize
 - ParameterNormalizer, 106
- depth
 - DirectTypes.h, 168
- dimension
 - DirectTypes.h, 168
- DIR
 - ConfigEditor, 43
- dirCoordinate
 - DirectTypes.h, 168
- direct, 19
 - level, 21
- DirectOptimizer, 57
 - activeReacts, 62
 - addActiveReacts, 59
 - D, 62
 - DirectOptimizer, 58
 - estimatedValue, 59
 - getName, 59
 - getPartitionSize, 60
 - getStatus, 60
 - getStatusBar, 60
 - getValues, 60
 - iterations, 62
 - level, 62
 - normalizer, 62
 - optimalRectangles, 61
 - removeActiveReacts, 61
 - runOptimization, 61
 - saveProgress, 61
 - stopCon, 63
 - trackProgress, 63
- DirectStoppingCondition, 63
 - DirectStoppingCondition, 64
 - evaluate, 65
 - NR_HYRECTS, 65
- DirectTypes.h
 - depth, 168
 - dimension, 168
 - dirCoordinate, 168
- DiscreteParameter, 65
 - DiscreteParameter, 67
 - getOffset, 68
 - getStep, 68
 - getTimes, 68
 - getVal, 68
 - offset, 69
 - setTimes, 68
 - setVal, 69
 - step, 69
 - times, 69
- divide
 - HyRect, 76
- editor
 - PlexeSimulationRunner, 112
- END_TIME
 - StoppingCondition, 135
- estimatedValue
 - DirectOptimizer, 59
- evaluate
 - Controller, 53
 - DirectStoppingCondition, 65
 - StoppingCondition, 133
 - StubController, 138
- Evaluation, 70
 - getName, 71
 - getStatus, 71
 - getStatusBar, 71
 - processOutput, 71, 72

- evaluation, 23
 - Controller, 55
- exec
 - CommandLine, 38
- f
 - StubController, 139
- functions
 - StubController, 139
- functionValue
 - Types.h, 202
- get
 - Controller::stepstate, 130
- get_constant_headway
 - constant_headway.py, 157
- getAvgValue
 - HyRect, 76
- getConditionFromJSON
 - StoppingCondition, 133
- getConfig
 - Parameter, 99
 - ParameterDefinition, 104
- getConfigByPath
 - Controller.cpp, 151
- getConfigPath
 - ConfigEditor, 41
- getControllerOption
 - ConfigEditor, 41
- getD
 - HyRect, 76
- getDepth
 - HyRect, 76
- getDiagonalLength
 - HyRect, 76
- getDir
 - ConfigEditor, 42
- getEpsilon
 - Levels, 82
- getIterationsSinceImprov
 - StoppingCondition, 134
- getLevel
 - Levels, 82
- getMax
 - Parameter, 99
 - ParameterDefinition, 104
- getMedian
 - ValueMap, 144
- getMin
 - Parameter, 99
 - ParameterDefinition, 104
- getName
 - ConstantHeadway, 45
 - DirectOptimizer, 59
 - Evaluation, 71
 - MonteCarlo, 87
 - Optimizer, 95
 - PlexeSimulationRunner, 110
 - RandomNeighbors, 117
 - SimulationRunner, 122
 - Status, 124
- getOffset
 - DiscreteParameter, 68
- getPartitionSize
 - DirectOptimizer, 60
- getPos
 - HyRect, 77
- getRectSubset
 - Levels, 82
- getResultPath
 - ConfigEditor, 42
- getRunId
 - PlexeSimulationRunner, 110
- getSamplingVertices
 - BaseRect, 31
 - ChildRect, 33
 - HyRect, 77
- getSize
 - ThreadsafeQueue< Key >, 140
 - ValueMap, 144
- getSplitDim
 - HyRect, 77
- getStartSize
 - ThreadsafeQueue< Key >, 141
- getStatus
 - ConstantHeadway, 46
 - DirectOptimizer, 60
 - Evaluation, 71
 - MonteCarlo, 87
 - Optimizer, 95
 - PlexeSimulationRunner, 110
 - RandomNeighbors, 117
 - SimulationRunner, 122
 - Status, 125
- getStatusBar
 - ConstantHeadway, 46
 - DirectOptimizer, 60
 - Evaluation, 71
 - MonteCarlo, 88
 - Optimizer, 96
 - PlexeSimulationRunner, 111
 - RandomNeighbors, 117
 - SimulationRunner, 122
 - Status, 125
- getStep
 - DiscreteParameter, 68
- getTimes
 - DiscreteParameter, 68
- getTopVals
 - ValueMap, 144
- getUnit
 - Parameter, 99
 - ParameterDefinition, 104
- getVal
 - ContinuousParameter, 49
 - DiscreteParameter, 68
 - Parameter, 100

- getValueMap
 - Controller, [53](#)
 - Optimizer, [96](#)
- getValues
 - DirectOptimizer, [60](#)
 - ValueMap, [145](#)
- global
 - Levels, [83](#)
- GrahamScan, [72](#)
 - scan, [73](#)
- hartman
 - StubController.cpp, [153](#)
- HyRect, [73](#)
 - avgValue, [80](#)
 - D, [80](#)
 - divide, [76](#)
 - getAvgValue, [76](#)
 - getD, [76](#)
 - getDepth, [76](#)
 - getDiagonalLength, [76](#)
 - getPos, [77](#)
 - getSamplingVertices, [77](#)
 - getSplitDim, [77](#)
 - HyRect, [75](#)
 - operator!=, [77](#)
 - operator<, [78](#)
 - operator<=, [78](#)
 - operator>, [79](#)
 - operator>=, [79](#)
 - operator==, [78](#)
 - pos, [80](#)
 - setAvgValue, [79](#)
 - t, [80](#)
- hyrect, [24](#)
 - position, [25](#)
- insert
 - ValueMap, [145](#)
- interruptHandler
 - main.cpp, [162](#)
- isGlobal
 - Levels, [83](#)
- isKnown
 - ValueMap, [145](#)
- isTopValue
 - ValueMap, [146](#)
- iterations
 - DirectOptimizer, [62](#)
 - MonteCarlo, [88](#)
 - RandomNeighbors, [118](#)
- iterationsSinceImprov
 - StoppingCondition, [135](#)
- keepFiles
 - Controller, [55](#)
- L0_EPSILON
 - Levels, [84](#)
- L0_SIZE
 - Levels, [84](#)
- L1_EPSILON
 - Levels, [84](#)
- L1_SIZE
 - Levels, [84](#)
- L2_EPSILON
 - Levels, [84](#)
- L2_SIZE
 - Levels, [84](#)
- L3_EPSILON
 - Levels, [84](#)
- L3_SIZE
 - Levels, [85](#)
- LARGE_DIVIDER
 - StatusBar, [128](#)
- lastEvaluations
 - MonteCarlo, [89](#)
 - RandomNeighbors, [118](#)
- lastLocal
 - RandomNeighbors, [118](#)
- lastStatus
 - StatusBar, [128](#)
- lastStep
 - StatusBar, [129](#)
- lastVal
 - StatusBar, [129](#)
- level
 - direct, [21](#)
 - DirectOptimizer, [62](#)
 - Levels, [80](#)
 - currentLevel, [83](#)
 - getEpsilon, [82](#)
 - getLevel, [82](#)
 - getRectSubset, [82](#)
 - global, [83](#)
 - isGlobal, [83](#)
 - L0_EPSILON, [84](#)
 - L0_SIZE, [84](#)
 - L1_EPSILON, [84](#)
 - L1_SIZE, [84](#)
 - L2_EPSILON, [84](#)
 - L2_SIZE, [84](#)
 - L3_EPSILON, [84](#)
 - L3_SIZE, [85](#)
 - nextLevel, [83](#)
 - setGlobal, [83](#)
- localSearchProbability
 - RandomNeighbors, [118](#)
- lowerValues
 - ValueMap, [146](#)
- main
 - main.cpp, [162](#)
- main.cpp
 - ctr, [162](#)
 - interruptHandler, [162](#)
 - main, [162](#)
- max

- ParameterDefinition, 105
- min
 - ParameterDefinition, 105
- mins
 - StoppingCondition, 135
- MonteCarlo, 85
 - getName, 87
 - getStatus, 87
 - getStatusBar, 88
 - iterations, 88
 - lastEvaluations, 89
 - MonteCarlo, 87
 - parallelTrials, 89
 - runOptimization, 88
 - saveProgress, 88
 - stopCon, 89
 - trackProgress, 89
- montecarlo, 21
- Multithreaded
 - Multithreaded< Key, T, Compare, Allocator >, 91
- multithreaded
 - constant_headway.py, 157
- Multithreaded< Key, T, Compare, Allocator >, 89
 - Multithreaded, 91
 - multithreadFunction, 91
 - NR_THREADS, 93
 - queue, 93
 - runMultithreadedFunctions, 92
 - work, 92
- multithreadFunction
 - Multithreaded< Key, T, Compare, Allocator >, 91
- neighborhoodWidth
 - RandomNeighbors, 119
- next
 - Controller::stepstate, 130
- nextLevel
 - Levels, 83
- NO_NAME
 - Status, 125
- NO_STATUS_SUPPORT
 - Status, 125
- normalize
 - ParameterNormalizer, 107
- normalizer
 - DirectOptimizer, 62
- NR_ACCURACY_ITERATIONS
 - StoppingCondition, 135
- NR_EVALUATIONS
 - StoppingCondition, 135
- NR_HYRECTS
 - DirectStoppingCondition, 65
- NR_THREADS
 - ConstantHeadway, 47
 - Multithreaded< Key, T, Compare, Allocator >, 93
- offset
 - DiscreteParameter, 69
- operationsLock
 - ValueMap, 147
- operator!=
 - HyRect, 77
 - Parameter, 100
- operator<
 - HyRect, 78
 - Parameter, 100
- operator<=
 - HyRect, 78
 - Parameter, 101
- operator>
 - HyRect, 79
 - Parameter, 101
- operator>=
 - HyRect, 79
 - Parameter, 102
- operator()
 - CmpPairVectorSharedParameterFunctionvalue, 35
 - CmpPtrFunctionvalue, 36
 - CmpSharedHyrect, 36
 - CmpVectorSharedParameter, 37
- operator==
 - ChildRect, 34
 - HyRect, 78
 - Parameter, 101
- optimalRectangles
 - DirectOptimizer, 61
- Optimizer, 93
 - controller, 97
 - getName, 95
 - getStatus, 95
 - getStatusBar, 96
 - getValueMap, 96
 - Optimizer, 95
 - parameters, 97
 - requestValues, 96
 - runOptimization, 96
- optimizer, 23
 - Controller, 55
- parallelTrials
 - MonteCarlo, 89
 - RandomNeighbors, 119
- Parameter, 97
 - definition, 102
 - getConfig, 99
 - getMax, 99
 - getMin, 99
 - getUnit, 99
 - getVal, 100
 - operator!=, 100
 - operator<, 100
 - operator<=, 101
 - operator>, 101
 - operator>=, 102
 - operator==, 101
 - Parameter, 99
 - setVal, 102
- parameterCombination

- Types.h, 202
- ParameterDefinition, 102
 - config, 105
 - getConfig, 104
 - getMax, 104
 - getMin, 104
 - getUnit, 104
 - max, 105
 - min, 105
 - ParameterDefinition, 103
 - unit, 105
- ParameterNormalizer, 105
 - denormalize, 106
 - normalize, 107
 - ParameterNormalizer, 106
 - parameters, 107
- parameters, 25
 - Optimizer, 97
 - ParameterNormalizer, 107
- parent
 - ChildRect, 34
- pFunc
 - PythonScript, 114
- plex, 22
- PlexeSimulationRunner, 107
 - editor, 112
 - getName, 110
 - getRunId, 110
 - getStatus, 110
 - getStatusBar, 111
 - PlexeSimulationRunner, 109
 - REPEAT, 112
 - runNumber, 112
 - runNumberLock, 112
 - SCENARIOS, 112
 - work, 111
- pModule
 - PythonScript, 114
- pop
 - ThreadsafeQueue< Key >, 141
- pos
 - HyRect, 80
- position
 - hyrect, 25
- printResult
 - StatusBar, 127
- printResults
 - StatusBar, 127
- printStatus
 - StatusBar, 127
- printValues
 - Controller, 56
- processOutput
 - ConstantHeadway, 46, 47
 - Evaluation, 71, 72
- push
 - ThreadsafeQueue< Key >, 141
- PythonScript, 112
 - ~PythonScript, 114
 - pFunc, 114
 - pModule, 114
 - PythonScript, 113
- query
 - ValueMap, 146
- queue
 - Multithreaded< Key, T, Compare, Allocator >, 93
- queueLock
 - ThreadsafeQueue< Key >, 141
- RandomNeighbors, 114
 - getName, 117
 - getStatus, 117
 - getStatusBar, 117
 - iterations, 118
 - lastEvaluations, 118
 - lastLocal, 118
 - localSearchProbability, 118
 - neighborhoodWidth, 119
 - parallelTrials, 119
 - RandomNeighbors, 116
 - runOptimization, 117
 - saveProgress, 118
 - stopCon, 119
 - trackProgress, 119
- randomneighbors, 21
- removeActiveRects
 - DirectOptimizer, 61
- removeOldResultfiles
 - Controller, 54
 - StubController, 138
- REPEAT
 - PlexeSimulationRunner, 112
- replaceOption
 - ConfigEditor, 42
- requestValues
 - Controller, 54
 - Optimizer, 96
- RESULTS
 - ConfigEditor, 43
- run
 - Controller, 54
- runId
 - Types.h, 202
- runMultithreadedFunctions
 - Multithreaded< Key, T, Compare, Allocator >, 92
- runner, 26
 - Controller, 56
- runNumber
 - PlexeSimulationRunner, 112
- runNumberLock
 - PlexeSimulationRunner, 112
- runOptimization
 - DirectOptimizer, 61
 - MonteCarlo, 88
 - Optimizer, 96
 - RandomNeighbors, 117

- runSimulations
 - Controller, [54](#)
 - SimulationRunner, [123](#)
 - StubController, [139](#)
- safeQueue
 - ThreadsafeQueue< Key >, [142](#)
- saveProgress
 - DirectOptimizer, [61](#)
 - MonteCarlo, [88](#)
 - RandomNeighbors, [118](#)
- saveValues
 - Controller, [55](#)
- scan
 - GrahamScan, [73](#)
- SCENARIOS
 - PlexeSimulationRunner, [112](#)
- secureValue
 - ConstantHeadway, [47](#)
- setAvgValue
 - HyRect, [79](#)
- setGlobal
 - Levels, [83](#)
- setResultFiles
 - ConfigEditor, [43](#)
- setStartNow
 - StoppingCondition, [134](#)
- setTimes
 - DiscreteParameter, [68](#)
- setVal
 - ContinuousParameter, [50](#)
 - DiscreteParameter, [69](#)
 - Parameter, [102](#)
- shekel
 - StubController.cpp, [153](#)
- SimulationRunner, [119](#)
 - getName, [122](#)
 - getStatus, [122](#)
 - getStatusBar, [122](#)
 - runSimulations, [123](#)
 - SimulationRunner, [122](#)
 - work, [123](#)
- SMALL_DIVIDER
 - StatusBar, [129](#)
- startSize
 - ThreadsafeQueue< Key >, [142](#)
- Status, [123](#)
 - getName, [124](#)
 - getStatus, [125](#)
 - getStatusBar, [125](#)
 - NO_NAME, [125](#)
 - NO_STATUS_SUPPORT, [125](#)
- status, [25](#)
 - step, [26](#)
- StatusBar, [126](#)
 - LARGE_DIVIDER, [128](#)
 - lastStatus, [128](#)
 - lastStep, [129](#)
 - lastVal, [129](#)
 - printResult, [127](#)
 - printResults, [127](#)
 - printStatus, [127](#)
 - SMALL_DIVIDER, [129](#)
 - updateStatus, [128](#)
- statusBar
 - Controller, [56](#)
- statusInterval
 - Controller, [56](#)
- step
 - DiscreteParameter, [69](#)
 - status, [26](#)
- stepChanged
 - Controller::stepstate, [130](#)
- stepState
 - controller, [19](#)
- stopCon
 - DirectOptimizer, [63](#)
 - MonteCarlo, [89](#)
 - RandomNeighbors, [119](#)
- StoppingCondition, [131](#)
 - ACCURACY, [134](#)
 - bestVal, [135](#)
 - END_TIME, [135](#)
 - evaluate, [133](#)
 - getConditionFromJSON, [133](#)
 - getIterationsSinceImprov, [134](#)
 - iterationsSinceImprov, [135](#)
 - mins, [135](#)
 - NR_ACCURACY_ITERATIONS, [135](#)
 - NR_EVALUATIONS, [135](#)
 - setStartNow, [134](#)
 - StoppingCondition, [132](#), [133](#)
 - time_eval, [136](#)
 - updateAccuracy, [134](#)
- StubController, [136](#)
 - evaluate, [138](#)
 - f, [139](#)
 - functions, [139](#)
 - removeOldResultfiles, [138](#)
 - runSimulations, [139](#)
 - StubController, [138](#)
 - updateStatus, [139](#)
- StubController.cpp
 - hartman, [153](#)
 - shekel, [153](#)
- t
 - HyRect, [80](#)
- tba
 - ValueMap, [147](#)
- ThreadsafeQueue< Key >, [140](#)
 - getSize, [140](#)
 - getStartSize, [141](#)
 - pop, [141](#)
 - push, [141](#)
 - queueLock, [141](#)
 - safeQueue, [142](#)
 - startSize, [142](#)

- time_eval
 - StoppingCondition, [136](#)
- times
 - DiscreteParameter, [69](#)
- topEntries
 - ValueMap, [147](#)
- topResults
 - Controller, [56](#)
- topVals
 - ValueMap, [147](#)
- trackProgress
 - DirectOptimizer, [63](#)
 - MonteCarlo, [89](#)
 - RandomNeighbors, [119](#)
- Types.h
 - coordinate, [201](#)
 - functionValue, [202](#)
 - parameterCombination, [202](#)
 - runId, [202](#)
- unit
 - ParameterDefinition, [105](#)
- updateAccuracy
 - StoppingCondition, [134](#)
- updateMap
 - ValueMap, [146](#)
- updateStatus
 - Controller, [55](#)
 - StatusBar, [128](#)
 - StubController, [139](#)
- upperValues
 - ValueMap, [147](#)
- usedThreads
 - ConstantHeadway, [47](#)
- utils, [27](#)
- val
 - ContinuousParameter, [50](#)
- ValueMap, [142](#)
 - addValue, [144](#)
 - getMedian, [144](#)
 - getSize, [144](#)
 - getTopVals, [144](#)
 - getValues, [145](#)
 - insert, [145](#)
 - isKnown, [145](#)
 - isTopValue, [146](#)
 - lowerValues, [146](#)
 - operationsLock, [147](#)
 - query, [146](#)
 - tba, [147](#)
 - topEntries, [147](#)
 - topVals, [147](#)
 - updateMap, [146](#)
 - upperValues, [147](#)
 - ValueMap, [143](#)
 - values, [147](#)
- valueMap
 - Controller, [56](#)
- values
 - ValueMap, [147](#)
- work
 - Multithreaded< Key, T, Compare, Allocator >, [92](#)
 - PlexeSimulationRunner, [111](#)
 - SimulationRunner, [123](#)