

Simopticon

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 Optimization	3
1.4 Extension	4
1.4.1 Development	4
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	20
7.2.2.1 level	21
7.3 plexe	21
7.3.1 Detailed Description	21
7.4 constant_headway	21
7.4.1 Detailed Description	22
7.5 parameters	22
7.5.1 Detailed Description	22
7.6 evaluation	22
7.6.1 Detailed Description	23

7.7 runner	23
7.7.1 Detailed Description	23
7.8 status	23
7.8.1 Detailed Description	24
7.8.2 Enumeration Type Documentation	24
7.8.2.1 step	24
7.9 optimizer	24
7.9.1 Detailed Description	24
7.10 hyrect	24
7.10.1 Detailed Description	25
7.10.2 Enumeration Type Documentation	25
7.10.2.1 position	25
7.11 utils	25
7.11.1 Detailed Description	26
8 Class Documentation	27
8.1 Abortable Class Reference	27
8.1.1 Detailed Description	28
8.1.2 Member Function Documentation	28
8.1.2.1 abort()	28
8.1.3 Member Data Documentation	28
8.1.3.1 aborted	28
8.2 BaseRect Class Reference	28
8.2.1 Detailed Description	29
8.2.2 Constructor & Destructor Documentation	29
8.2.2.1 BaseRect()	29
8.2.3 Member Function Documentation	29
8.2.3.1 getSamplingVertices()	29
8.3 ChildRect Class Reference	30
8.3.1 Detailed Description	31
8.3.2 Constructor & Destructor Documentation	31
8.3.2.1 ChildRect()	31
8.3.3 Member Function Documentation	31
8.3.3.1 getSamplingVertices()	31
8.3.3.2 operator==()	32
8.3.4 Member Data Documentation	32
8.3.4.1 parent	32
8.4 CmpPairVectorSharedParameterFunctionvalue Struct Reference	32
8.4.1 Detailed Description	33
8.4.2 Member Function Documentation	33
8.4.2.1 operator()()	33
8.5 CmpPtrFunctionvalue Struct Reference	33

8.5.1 Detailed Description	33
8.5.2 Member Function Documentation	33
8.5.2.1 operator()()	34
8.6 CmpSharedHyrect Struct Reference	34
8.6.1 Detailed Description	34
8.6.2 Member Function Documentation	34
8.6.2.1 operator()()	34
8.7 CmpVectorSharedParameter Struct Reference	35
8.7.1 Detailed Description	35
8.7.2 Member Function Documentation	35
8.7.2.1 operator()()	35
8.8 CommandLine Class Reference	36
8.8.1 Detailed Description	36
8.8.2 Member Function Documentation	36
8.8.2.1 exec()	36
8.9 ConfigEditor Class Reference	37
8.9.1 Detailed Description	38
8.9.2 Constructor & Destructor Documentation	38
8.9.2.1 ConfigEditor()	38
8.9.3 Member Function Documentation	38
8.9.3.1 createConfig()	38
8.9.3.2 deleteConfig()	38
8.9.3.3 getConfigPath()	39
8.9.3.4 getControllerOption()	39
8.9.3.5 getDir()	39
8.9.3.6 getResultPath()	40
8.9.3.7 replaceOption() [1/2]	40
8.9.3.8 replaceOption() [2/2]	40
8.9.3.9 setResultFiles()	40
8.9.4 Member Data Documentation	41
8.9.4.1 CONFIG	41
8.9.4.2 CONTROLLER	41
8.9.4.3 DIR	41
8.9.4.4 RESULTS	41
8.10 ConstantHeadway Class Reference	41
8.10.1 Detailed Description	43
8.10.2 Constructor & Destructor Documentation	43
8.10.2.1 ConstantHeadway()	43
8.10.3 Member Function Documentation	43
8.10.3.1 getName()	43
8.10.3.2 getStatus()	44
8.10.3.3 getStatusBar()	44

8.10.3.4 processOutput() [1/2]	44
8.10.3.5 processOutput() [2/2]	44
8.10.3.6 secureValue()	45
8.10.4 Member Data Documentation	45
8.10.4.1 NR_THREADS	45
8.10.4.2 usedThreads	45
8.11 ContinuousParameter Class Reference	45
8.11.1 Detailed Description	46
8.11.2 Constructor & Destructor Documentation	47
8.11.2.1 ContinuousParameter() [1/2]	47
8.11.2.2 ContinuousParameter() [2/2]	47
8.11.3 Member Function Documentation	47
8.11.3.1 getVal()	47
8.11.3.2 setVal()	47
8.11.4 Member Data Documentation	49
8.11.4.1 val	49
8.12 Controller Class Reference	49
8.12.1 Detailed Description	51
8.12.2 Constructor & Destructor Documentation	51
8.12.2.1 Controller()	51
8.12.3 Member Function Documentation	52
8.12.3.1 abort()	52
8.12.3.2 evaluate()	52
8.12.3.3 getValueMap()	52
8.12.3.4 removeOldResultfiles()	52
8.12.3.5 requestValues()	53
8.12.3.6 run()	53
8.12.3.7 runSimulations()	53
8.12.3.8 saveValues()	53
8.12.3.9 updateStatus()	54
8.12.4 Member Data Documentation	54
8.12.4.1 evaluation	54
8.12.4.2 keepFiles	54
8.12.4.3 optimizer	54
8.12.4.4 printValues	54
8.12.4.5 runner	54
8.12.4.6 statusBar	55
8.12.4.7 statusInterval	55
8.12.4.8 topResults	55
8.12.4.9 valueMap	55
8.13 DirectOptimizer Class Reference	55
8.13.1 Detailed Description	57

8.13.2 Constructor & Destructor Documentation	57
8.13.2.1 DirectOptimizer()	58
8.13.3 Member Function Documentation	58
8.13.3.1 addActiveRects()	58
8.13.3.2 estimatedValue()	58
8.13.3.3 getName()	59
8.13.3.4 getPartitionSize()	59
8.13.3.5 getStatus()	59
8.13.3.6 getStatusBar()	59
8.13.3.7 getValues()	59
8.13.3.8 optimalRectangles()	60
8.13.3.9 removeActiveRects()	60
8.13.3.10 runOptimization()	60
8.13.3.11 saveProgress()	61
8.13.4 Member Data Documentation	61
8.13.4.1 activeRects	61
8.13.4.2 D	61
8.13.4.3 iterations	61
8.13.4.4 level	61
8.13.4.5 normalizer	61
8.13.4.6 stopCon	62
8.13.4.7 trackProgress	62
8.14 DiscreteParameter Class Reference	62
8.14.1 Detailed Description	63
8.14.2 Constructor & Destructor Documentation	63
8.14.2.1 DiscreteParameter() [1/2]	63
8.14.2.2 DiscreteParameter() [2/2]	64
8.14.3 Member Function Documentation	64
8.14.3.1 getOffset()	64
8.14.3.2 getStep()	64
8.14.3.3 getTimes()	64
8.14.3.4 getVal()	65
8.14.3.5 setTimes()	65
8.14.3.6 setVal()	65
8.14.4 Member Data Documentation	65
8.14.4.1 offset	65
8.14.4.2 step	66
8.14.4.3 times	66
8.15 Evaluation Class Reference	66
8.15.1 Detailed Description	67
8.15.2 Member Function Documentation	67
8.15.2.1 getName()	67

8.15.2.2	getStatus()	68
8.15.2.3	getStatusBar()	68
8.15.2.4	processOutput() [1/2]	68
8.15.2.5	processOutput() [2/2]	68
8.16	GrahamScan Class Reference	69
8.16.1	Detailed Description	69
8.16.2	Member Function Documentation	69
8.16.2.1	scan()	69
8.17	HyRect Class Reference	70
8.17.1	Detailed Description	71
8.17.2	Constructor & Destructor Documentation	71
8.17.2.1	HyRect()	72
8.17.3	Member Function Documentation	72
8.17.3.1	divide()	72
8.17.3.2	getAvgValue()	72
8.17.3.3	getD()	72
8.17.3.4	getDepth()	73
8.17.3.5	getDiagonalLength()	73
8.17.3.6	getPos()	73
8.17.3.7	getSamplingVertices()	73
8.17.3.8	getSplitDim()	74
8.17.3.9	operator!=(())	74
8.17.3.10	operator<()	74
8.17.3.11	operator<=()	74
8.17.3.12	operator==(())	75
8.17.3.13	operator>()	75
8.17.3.14	operator>=()	75
8.17.3.15	setAvgValue()	76
8.17.4	Member Data Documentation	76
8.17.4.1	avgValue	76
8.17.4.2	D	76
8.17.4.3	pos	76
8.17.4.4	t	77
8.18	Levels Class Reference	77
8.18.1	Detailed Description	78
8.18.2	Member Function Documentation	78
8.18.2.1	getEpsilon()	78
8.18.2.2	getLevel()	78
8.18.2.3	getRectSubset()	79
8.18.2.4	isGlobal()	79
8.18.2.5	nextLevel()	79
8.18.2.6	setGlobal()	79

8.18.3 Member Data Documentation	80
8.18.3.1 currentLevel	80
8.18.3.2 global	80
8.18.3.3 L0_EPSILON	80
8.18.3.4 L0_SIZE	80
8.18.3.5 L1_EPSILON	80
8.18.3.6 L1_SIZE	80
8.18.3.7 L2_EPSILON	80
8.18.3.8 L2_SIZE	81
8.18.3.9 L3_EPSILON	81
8.18.3.10 L3_SIZE	81
8.19 Multithreaded< Key, T, Compare, Allocator > Class Template Reference	81
8.19.1 Detailed Description	83
8.19.2 Constructor & Destructor Documentation	83
8.19.2.1 Multithreaded()	83
8.19.3 Member Function Documentation	83
8.19.3.1 multithreadFunction()	83
8.19.3.2 runMultithreadedFunctions()	84
8.19.3.3 work()	84
8.19.4 Member Data Documentation	84
8.19.4.1 NR_THREADS	85
8.19.4.2 queue	85
8.20 Optimizer Class Reference	85
8.20.1 Detailed Description	87
8.20.2 Constructor & Destructor Documentation	87
8.20.2.1 Optimizer()	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 getValueMap()	88
8.20.3.5 requestValues()	88
8.20.3.6 runOptimization()	88
8.20.4 Member Data Documentation	88
8.20.4.1 controller	89
8.20.4.2 parameters	89
8.21 Parameter Class Reference	89
8.21.1 Detailed Description	90
8.21.2 Constructor & Destructor Documentation	90
8.21.2.1 Parameter()	90
8.21.3 Member Function Documentation	90
8.21.3.1 getConfig()	91

8.21.3.2 getMax()	91
8.21.3.3 getMin()	91
8.21.3.4 getUnit()	91
8.21.3.5 getVal()	91
8.21.3.6 operator"!=()	92
8.21.3.7 operator<()	92
8.21.3.8 operator<=()	92
8.21.3.9 operator==()	93
8.21.3.10 operator>()	93
8.21.3.11 operator>=()	93
8.21.3.12 setVal()	94
8.21.4 Member Data Documentation	94
8.21.4.1 definition	94
8.22 ParameterDefinition Class Reference	94
8.22.1 Detailed Description	95
8.22.2 Constructor & Destructor Documentation	95
8.22.2.1 ParameterDefinition()	95
8.22.3 Member Function Documentation	95
8.22.3.1 getConfig()	95
8.22.3.2 getMax()	96
8.22.3.3 getMin()	96
8.22.3.4 getUnit()	96
8.22.4 Member Data Documentation	96
8.22.4.1 config	96
8.22.4.2 max	97
8.22.4.3 min	97
8.22.4.4 unit	97
8.23 ParameterNormalizer Class Reference	97
8.23.1 Detailed Description	98
8.23.2 Constructor & Destructor Documentation	98
8.23.2.1 ParameterNormalizer()	98
8.23.3 Member Function Documentation	98
8.23.3.1 denormalize()	98
8.23.3.2 normalize()	98
8.23.4 Member Data Documentation	99
8.23.4.1 parameters	99
8.24 PlexeSimulationRunner Class Reference	99
8.24.1 Detailed Description	101
8.24.2 Constructor & Destructor Documentation	101
8.24.2.1 PlexeSimulationRunner()	101
8.24.3 Member Function Documentation	101
8.24.3.1 getName()	101

8.24.3.2 getRunId()	102
8.24.3.3 getStatus()	102
8.24.3.4 getStatusBar()	102
8.24.3.5 work() [1/2]	102
8.24.3.6 work() [2/2]	103
8.24.4 Member Data Documentation	103
8.24.4.1 editor	103
8.24.4.2 REPEAT	103
8.24.4.3 runNumber	103
8.24.4.4 runNumberLock	104
8.24.4.5 SCENARIOS	104
8.25 PythonScript Class Reference	104
8.25.1 Detailed Description	105
8.25.2 Constructor & Destructor Documentation	105
8.25.2.1 PythonScript()	105
8.25.2.2 ~PythonScript()	105
8.25.3 Member Data Documentation	105
8.25.3.1 pFunc	105
8.25.3.2 pModule	105
8.26 SimulationRunner Class Reference	106
8.26.1 Detailed Description	107
8.26.2 Constructor & Destructor Documentation	108
8.26.2.1 SimulationRunner()	108
8.26.3 Member Function Documentation	108
8.26.3.1 getName()	108
8.26.3.2 getStatus()	108
8.26.3.3 getStatusBar()	108
8.26.3.4 runSimulations()	109
8.26.3.5 work()	109
8.27 Status Class Reference	109
8.27.1 Detailed Description	110
8.27.2 Member Function Documentation	110
8.27.2.1 getName()	111
8.27.2.2 getStatus()	111
8.27.2.3 getStatusBar()	111
8.27.3 Member Data Documentation	111
8.27.3.1 NO_NAME	111
8.27.3.2 NO_STATUS_SUPPORT	112
8.28 StatusBar Class Reference	112
8.28.1 Detailed Description	113
8.28.2 Member Function Documentation	113
8.28.2.1 printResult()	113

8.28.2.2 printResults()	113
8.28.2.3 printStatus()	113
8.28.2.4 updateStatus()	114
8.28.3 Member Data Documentation	114
8.28.3.1 LARGE_DIVIDER	114
8.28.3.2 lastStatus	114
8.28.3.3 lastStep	115
8.28.3.4 lastVal	115
8.28.3.5 SMALL_DIVIDER	115
8.29 Controller::stepstate Struct Reference	115
8.29.1 Detailed Description	116
8.29.2 Member Function Documentation	116
8.29.2.1 get()	116
8.29.2.2 next()	116
8.29.3 Member Data Documentation	116
8.29.3.1 currentStep	116
8.29.3.2 stepChanged	116
8.30 StoppingCondition Class Reference	117
8.30.1 Detailed Description	118
8.30.2 Constructor & Destructor Documentation	118
8.30.2.1 StoppingCondition() [1/2]	118
8.30.2.2 StoppingCondition() [2/2]	118
8.30.3 Member Function Documentation	118
8.30.3.1 evaluate()	118
8.30.3.2 getIterationsSinceImprov()	119
8.30.3.3 setStartNow()	119
8.30.3.4 updateAccuracy()	119
8.30.4 Member Data Documentation	120
8.30.4.1 ACCURACY	120
8.30.4.2 bestVal	120
8.30.4.3 END_TIME	120
8.30.4.4 iterationsSinceImprov	120
8.30.4.5 mins	120
8.30.4.6 NR_ACCURACY_ITERATIONS	120
8.30.4.7 NR_EVALUATIONS	121
8.30.4.8 NR_HYRECTS	121
8.30.4.9 time_eval	121
8.31 StubController Class Reference	121
8.31.1 Detailed Description	122
8.31.2 Constructor & Destructor Documentation	123
8.31.2.1 StubController()	123
8.31.3 Member Function Documentation	123

8.31.3.1 evaluate()	123
8.31.3.2 removeOldResultfiles()	123
8.31.3.3 runSimulations()	124
8.31.3.4 updateStatus()	124
8.31.4 Member Data Documentation	124
8.31.4.1 f	124
8.31.4.2 functions	124
8.32 ThreadsafeQueue< Key > Class Template Reference	125
8.32.1 Detailed Description	125
8.32.2 Member Function Documentation	125
8.32.2.1 getSize()	126
8.32.2.2 getStartSize()	126
8.32.2.3 pop()	126
8.32.2.4 push()	126
8.32.3 Member Data Documentation	126
8.32.3.1 queueLock	127
8.32.3.2 safeQueue	127
8.32.3.3 startSize	127
8.33 ValueMap Class Reference	127
8.33.1 Detailed Description	128
8.33.2 Constructor & Destructor Documentation	128
8.33.2.1 ValueMap()	129
8.33.3 Member Function Documentation	130
8.33.3.1 addValue()	130
8.33.3.2 getMedian()	130
8.33.3.3 getSize()	130
8.33.3.4 getTopVals()	130
8.33.3.5 getValues()	131
8.33.3.6 insert()	131
8.33.3.7 isKnown()	131
8.33.3.8 isTopValue()	132
8.33.3.9 query()	132
8.33.3.10 updateMap()	132
8.33.4 Member Data Documentation	132
8.33.4.1 lowerValues	133
8.33.4.2 operationsLock	133
8.33.4.3 tba	133
8.33.4.4 topEntries	133
8.33.4.5 topVals	133
8.33.4.6 upperValues	133
8.33.4.7 values	133

9 File Documentation	135
9.1 src/ComparisonFunctions.h File Reference	135
9.1.1 Detailed Description	136
9.2 src/controller/Controller.cpp File Reference	136
9.2.1 Detailed Description	136
9.2.2 Function Documentation	136
9.2.2.1 getConfigByPath()	136
9.3 src/controller/Controller.h File Reference	137
9.3.1 Detailed Description	138
9.4 src/controller/StubController.cpp File Reference	138
9.4.1 Detailed Description	138
9.4.2 Function Documentation	138
9.4.2.1 hartman()	138
9.4.2.2 shekel()	139
9.5 src/controller/StubController.h File Reference	139
9.5.1 Detailed Description	140
9.6 src/controller/ValueMap.cpp File Reference	140
9.6.1 Detailed Description	140
9.7 src/controller/ValueMap.h File Reference	140
9.7.1 Detailed Description	141
9.8 src/evaluation/constant_headway/constant_headway.py File Reference	141
9.8.1 Detailed Description	142
9.8.2 Function Documentation	142
9.8.2.1 get_constant_headway()	142
9.8.2.2 multithreaded()	142
9.9 src/evaluation/constant_headway/ConstantHeadway.cpp File Reference	143
9.9.1 Detailed Description	143
9.10 src/evaluation/constant_headway/ConstantHeadway.h File Reference	143
9.10.1 Detailed Description	144
9.11 src/evaluation/Evaluation.cpp File Reference	144
9.11.1 Detailed Description	145
9.12 src/evaluation/Evaluation.h File Reference	145
9.12.1 Detailed Description	146
9.13 src/main.cpp File Reference	146
9.13.1 Detailed Description	147
9.13.2 Function Documentation	147
9.13.2.1 interruptHandler()	147
9.13.2.2 main()	147
9.13.3 Variable Documentation	147
9.13.3.1 ctr	147
9.14 src/optimizer/direct/DirectComparisonFunctions.h File Reference	147
9.14.1 Detailed Description	148

9.15 src/optimizer/direct/DirectOptimizer.cpp File Reference	148
9.15.1 Detailed Description	149
9.16 src/optimizer/direct/DirectOptimizer.h File Reference	149
9.16.1 Detailed Description	150
9.17 src/optimizer/direct/DirectTypes.h File Reference	150
9.17.1 Detailed Description	150
9.17.2 Typedef Documentation	151
9.17.2.1 depth	151
9.17.2.2 dimension	151
9.17.2.3 dirCoordinate	151
9.18 src/optimizer/direct/GrahamScan.cpp File Reference	151
9.18.1 Detailed Description	151
9.19 src/optimizer/direct/GrahamScan.h File Reference	151
9.19.1 Detailed Description	152
9.20 src/optimizer/direct/hyrect/BaseRect.cpp File Reference	152
9.20.1 Detailed Description	153
9.21 src/optimizer/direct/hyrect/BaseRect.h File Reference	153
9.21.1 Detailed Description	154
9.22 src/optimizer/direct/hyrect/ChildRect.cpp File Reference	154
9.22.1 Detailed Description	154
9.23 src/optimizer/direct/hyrect/ChildRect.h File Reference	155
9.23.1 Detailed Description	155
9.24 src/optimizer/direct/hyrect/HyRect.cpp File Reference	155
9.24.1 Detailed Description	156
9.25 src/optimizer/direct/hyrect/HyRect.h File Reference	156
9.25.1 Detailed Description	157
9.26 src/optimizer/direct/Levels.cpp File Reference	157
9.26.1 Detailed Description	158
9.27 src/optimizer/direct/Levels.h File Reference	158
9.27.1 Detailed Description	159
9.28 src/optimizer/direct/ParameterNormalizer.cpp File Reference	159
9.28.1 Detailed Description	160
9.29 src/optimizer/direct/ParameterNormalizer.h File Reference	160
9.29.1 Detailed Description	161
9.30 src/optimizer/direct/StoppingCondition.cpp File Reference	161
9.30.1 Detailed Description	162
9.30.2 Function Documentation	162
9.30.2.1 getConditionFromJSON()	162
9.31 src/optimizer/direct/StoppingCondition.h File Reference	162
9.31.1 Detailed Description	163
9.32 src/optimizer/Optimizer.cpp File Reference	163
9.32.1 Detailed Description	164

9.33 src/optimizer/Optimizer.h File Reference	164
9.33.1 Detailed Description	165
9.34 src/parameters/ContinuousParameter.cpp File Reference	165
9.34.1 Detailed Description	165
9.35 src/parameters/ContinuousParameter.h File Reference	165
9.35.1 Detailed Description	166
9.36 src/parameters/DiscreteParameter.cpp File Reference	166
9.36.1 Detailed Description	167
9.37 src/parameters/DiscreteParameter.h File Reference	167
9.37.1 Detailed Description	168
9.38 src/parameters/Parameter.cpp File Reference	168
9.38.1 Detailed Description	169
9.39 src/parameters/Parameter.h File Reference	169
9.39.1 Detailed Description	170
9.40 src/parameters/ParameterDefinition.cpp File Reference	170
9.40.1 Detailed Description	171
9.41 src/parameters/ParameterDefinition.h File Reference	171
9.41.1 Detailed Description	172
9.42 src/runner/plex/ConfigEditor.cpp File Reference	172
9.42.1 Detailed Description	172
9.43 src/runner/plex/ConfigEditor.h File Reference	173
9.43.1 Detailed Description	173
9.44 src/runner/plex/PlexSimulationRunner.cpp File Reference	174
9.44.1 Detailed Description	174
9.45 src/runner/plex/PlexSimulationRunner.h File Reference	174
9.45.1 Detailed Description	175
9.46 src/runner/SimulationRunner.cpp File Reference	175
9.46.1 Detailed Description	175
9.47 src/runner/SimulationRunner.h File Reference	175
9.47.1 Detailed Description	176
9.48 src/status/Status.cpp File Reference	176
9.48.1 Detailed Description	177
9.49 src/status/Status.h File Reference	177
9.49.1 Detailed Description	178
9.50 src/status/StatusBar.cpp File Reference	178
9.50.1 Detailed Description	178
9.51 src/status/StatusBar.h File Reference	178
9.51.1 Detailed Description	179
9.52 src/Types.h File Reference	179
9.52.1 Detailed Description	180
9.52.2 Typedef Documentation	180
9.52.2.1 coordinate	180

9.52.2.2 functionValue	181
9.52.2.3 parameterCombination	181
9.52.2.4 runId	181
9.53 src/Utils/Abortable.cpp File Reference	181
9.53.1 Detailed Description	181
9.54 src/Utils/Abortable.h File Reference	181
9.54.1 Detailed Description	182
9.55 src/Utils/CommandLine.cpp File Reference	182
9.55.1 Detailed Description	182
9.56 src/Utils/CommandLine.h File Reference	183
9.56.1 Detailed Description	183
9.57 src/Utils/Multithreaded.h File Reference	183
9.57.1 Detailed Description	184
9.58 src/Utils/Multithreaded.hpp File Reference	184
9.58.1 Detailed Description	185
9.59 src/Utils/PythonScript.cpp File Reference	185
9.59.1 Detailed Description	186
9.60 src/Utils/PythonScript.h File Reference	186
9.60.1 Detailed Description	187
9.61 src/Utils/ThreadSafeQueue.h File Reference	187
9.61.1 Detailed Description	188
9.62 src/Utils/ThreadSafeQueue.hpp File Reference	188
9.62.1 Detailed Description	189
Index	191

Chapter 1

Documentation

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

1.1 Overview

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

The described process is distributed over four major components:

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

Extensions of the framework may introduce new **Optimizer**, **SimulationRunner** and **Evaluation** implementations (see [Extension](#)). Currently, there is only one implementation of each component, tailored for the optimization of platoon controllers using the **Plexe** framework.

The full API documentation may be found on peternaggschga.github.io/simopticon 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/plexe.json`. There you have to set the `configDirectory` key to match the path to the directory containing your Plexe configuration (`omnetpp.ini`). For default installations that should be something along the lines of `[installation-directory]/plexe/examples/platooning`.

ConstantHeadway

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

1.3.2 Optimization

The optimization is invoked on the command line by executing the program built in [Setup](#). The call on the command line has one mandatory and one optional argument. The First argument must be the path to the main config, i.e. `config/simopticon.json`. A valid call to an optimization could be:

```
./simopticon ../config/simopticon.json
```

If a second argument is given, instead of running actual simulations with the configured [SimulationRunner](#) and evaluating their results with an [Evaluation](#), the [StubController](#) is used. [StubController](#) can be used to implement and

optimize benchmark functions to test [Optimizer](#) implementations without relying on actual costly simulations. The second argument holds the name of the function to be optimized, i.e., one of the following:

- quadratic (squares all [Parameter](#) values and adds them up)
- `branin`
- `goldprice`
- `camel6`
- `shubert`
- `hartman3`
- `shekel5`
- `shekel7`
- `shekel10`
- `hartman6`

A valid call to the optimization of a benchmark function could be:

```
./simopticon ../config/simopticon.json branin
```

Please note that you need to define the optimized parameters in `config/simopticon.json` even when you are optimizing a benchmark.

1.4 Extension

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

1.4.1 Development

When developing new implementations of components, please stick to the project structure — [Optimizer](#) extensions go into `src/optimizer`, [SimulationRunner](#) extensions go into `src/runner` and [Evaluation](#) extensions go into `src/evaluation`. If your implementation needs a more sophisticated implementation of the [Parameter](#) class than the ones provided in `src/parameters`, feel free to extend the abstract [Parameter](#) class.

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

The `src/Types.h` header file defines framework-wide types such as `functionValue` for values returned by the [Evaluation](#) component or `coordinate` which is used to store [Parameter](#) values. The `src/ComparisonFunctions.h` header file defines comparison functions, which can be used in STL containers that are ordered. E.g. `CmpVectorSharedParameter` can be used to compare two objects of type `vector<shared_ptr<Parameter>>`.

Optimization Strategies

To add a new optimization strategy, you have to extend the [Optimizer](#) class. You need to override the [Optimizer::runOptimization](#) method which should start the optimization process and only return when your strategy is finished or if the [Optimizer::abort](#) method is called which you should implement too.

[Optimizer](#) extensions can instruct the [Controller](#) to start simulations and evaluate them with the [Optimizer::requestValues](#) method. Please try to commission as many Parameters as possible in one call of the method so the other components may parallelize calculations.

Please consider overriding the methods provided by the [Status](#) interface to give the user a sense of what is happening.

Simulation Execution

To add a new way of executing simulations, you have to extend the `SimulationRunner` class. You need to override the `SimulationRunner::work` function, which is run concurrently for all `Parameter` vectors provided to `SimulationRunner::runSimulations`. If you want to prohibit concurrent execution, you may override `SimulationRunner::runSimulations` instead (in that case, `SimulationRunner::work` should return an empty pair). See documentation of `Multithreaded` class for more information on that.

`SimulationRunner::work` should run a simulation with the given parameters and return a path to the result files and a set of identifiers relating to simulation runs. The interface for the identifiers is very loosely defined — if your `Evaluation` does not need any identifiers of simulation runs, you may return an empty set. Please be aware that the `Controller` might try to delete the path you return after some time, so that should not be an empty path! Other than that, it is not further standardized what must be returned as a path and identifiers as long as your `Evaluation` component can evaluate the simulation based on the returned information.

Please consider overriding the methods provided by the `Status` interface to give the user a sense of what is happening.

Simulation Evaluation

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

Please consider overriding the methods provided by the `Status` interface to give the user a sense of what is happening.

1.4.2 Integration

All newly added classes must be registered in `CMakeList.txt` so the compiler does not ignore them! External dependencies and added libraries should be included there too.

To make your new component available for configuration, you must add it to the constructor of the `Controller` class. Let's assume you wrote a new `Optimizer` implementation. First you need to create a JSON configuration file in `config/optimizer`. There you can define any desired options for your component.

The next step is editing the `Controller` class to make your `Optimizer` available. To do that, you find the "Optimizer settings" in the constructor of the `Controller`. There you add another case to the `if`-Statement where `opt` equals the name of your component (this is the name that will be set in the main config later, see [Configuration](#)). In the added case you can read the necessary options from the JSON object in `optimizerConfig`. You have to set `Controller::optimizer` to an `unique_ptr<Optimizer>`, owning a new instance of your `Optimizer` implementation.

When this setup is complete, you may build the framework again and update the main configuration to use your new `Optimizer` by changing the `optimizer.optimizer` key to the name of your `Optimizer` and the `optimizer.config` key to the path of your created JSON configuration file.

Chapter 2

Todo List

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

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

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

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Abortable	27
Controller	49
StubController	121
Optimizer	85
DirectOptimizer	55
CmpPairVectorSharedParameterFunctionvalue	32
CmpPtrFunctionvalue	33
CmpSharedHyrect	34
CmpVectorSharedParameter	35
CommandLine	36
ConfigEditor	37
GrahamScan	69
HyRect	70
BaseRect	28
ChildRect	30
Levels	77
Multithreaded< Key, T, Compare, Allocator >	81
Multithreaded< parameterCombination, std::pair< std::filesystem::path, std::set< runId > >, Cmp← VectorSharedParameter >	81
SimulationRunner	106
PlexeSimulationRunner	99
Multithreaded< std::pair< std::filesystem::path, std::pair< std::string, unsigned int > >, bool >	81
PlexeSimulationRunner	99
Parameter	89
ContinuousParameter	45
DiscreteParameter	62
ParameterDefinition	94
ParameterNormalizer	97
PythonScript	104
ConstantHeadway	41
Status	109
Evaluation	66
ConstantHeadway	41
Optimizer	85
SimulationRunner	106
StatusBar	112
Controller::stepstate	115
StoppingCondition	117

ThreadsafeQueue< Key >	125
ValueMap	127

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	27
BaseRect	A class representing a HyRect without a parent rectangle	28
ChildRect	A class representing a HyRect that has a parent HyRect	30
CmpPairVectorSharedParameterFunctionvalue	This struct implements the comparison of two pairs of parameterCombination and function value	32
CmpPtrFunctionvalue	This struct implements the comparison of two pointers to function values	33
CmpSharedHyrect	This struct implements the comparison of two shared pointers to HyRect instances	34
CmpVectorSharedParameter	This struct implements the comparison of two vectors of Parameter references	35
CommandLine	A class containing functionality for executing commands on UNIX shell	36
ConfigEditor	A class capable of creating <code>.ini</code> files with certain options based on a complete <code>omnetpp.ini</code>	37
ConstantHeadway	A wrapper for the <code>constant_headway.py</code> script	41
ContinuousParameter	Implements a Parameter using continuous values in the form of floating point numbers	45
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	49
DirectOptimizer	A class capable of finding the minimum of a blackbox function using the DIRECT algorithm	55
DiscreteParameter	Implements a Parameter using discrete values	62
Evaluation	A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization	66
GrahamScan	A class providing functionality for finding the lower right convex hull of a set of points	69
HyRect	An abstract class representing a rectangular part of the search space	70
Levels	A providing functionality for the usage of different weightings between local and global search throughout the optimization using different levels	77

Multithreaded< Key, T, Compare, Allocator >	
A class implementing concurrent execution of the same function for different arguments	81
Optimizer	
A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs	85
Parameter	
A class acting as the container of the value of a parameter defined by a ParameterDefinition .	89
ParameterDefinition	
A class storing information on the properties of parameters that are being optimized	94
ParameterNormalizer	
A class used for transforming parameters between the actual Parameter space and the unit hypercube used in DIRECT algorithm	97
PlexeSimulationRunner	
A class capable of starting platooning simulations in the Plexe framework with given parameterCombinations	99
PythonScript	
A class containing functionality for interfacing with the function of a Python module on creation	104
SimulationRunner	
A class capable of running simulations with certain parameterCombinations	106
Status	
An interface defining functions for status updates on configuration and progress of a class . . .	109
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	112
Controller::stepstate	
A struct keeping track of the currently running optimization step for StatusBar::updateStatus . .	115
StoppingCondition	
A class used for deciding whether the DIRECT should be stopped	117
StubController	
A class that mocks behaviour of Controller	121
ThreadsafeQueue< Key >	
A container class of a queue that is safe for concurrent access of different threads	125
ValueMap	
A container managing a map data structure that maps parameterCombinations to their respective found values	127

Chapter 6

File Index

6.1 File List

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

src/ ComparisonFunctions.h	
In this file, comparison functions are defined which should be used across the whole framework	135
src/ main.cpp	
In this file, the main function running the <i>Simopticon</i> framework is defined	146
src/ Types.h	
In this file, types are defined which should be used across the whole framework	179
src/controller/ Controller.cpp	
In this file, the implementation of the Controller class is defined	136
src/controller/ Controller.h	
In this file, the header of the Controller class is defined	137
src/controller/ StubController.cpp	
In this file, the implementation of the StubController class is defined	138
src/controller/ StubController.h	
In this file, the header of the StubController class is defined	139
src/controller/ ValueMap.cpp	
In this file, the implementation of the ValueMap class is defined	140
src/controller/ ValueMap.h	
In this file, the header of the ValueMap class is defined	140
src/evaluation/ Evaluation.cpp	
In this file, the implementation of the Evaluation class is defined	144
src/evaluation/ Evaluation.h	
In this file, the header of the Evaluation class is defined	145
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	141
src/evaluation/constant_headway/ ConstantHeadway.cpp	
In this file, the implementation of the ConstantHeadway class is defined	143
src/evaluation/constant_headway/ ConstantHeadway.h	
In this file, the header of the ConstantHeadway class is defined	143
src/optimizer/ Optimizer.cpp	
In this file, the implementation of the Optimizer class is defined	163
src/optimizer/ Optimizer.h	
In this file, the header of the Optimizer class is defined	164
src/optimizer/direct/ DirectComparisonFunctions.h	
In this file, comparison functions are defined which are used in the direct module	147
src/optimizer/direct/ DirectOptimizer.cpp	
In this file, the implementation of the DirectOptimizer class is defined	148
src/optimizer/direct/ DirectOptimizer.h	
In this file, the header of the DirectOptimizer class is defined	149

src/optimizer/direct/ DirectTypes.h	
In this file, types are defined which are used in the direct module	150
src/optimizer/direct/ GrahamScan.cpp	
In this file, the implementation of the GrahamScan class is defined	151
src/optimizer/direct/ GrahamScan.h	
In this file, the header of the GrahamScan class is defined	151
src/optimizer/direct/ Levels.cpp	
In this file, the implementation of the Levels class is defined	157
src/optimizer/direct/ Levels.h	
In this file, the header of the Levels class is defined	158
src/optimizer/direct/ ParameterNormalizer.cpp	
In this file, the implementation of the ParameterNormalizer class is defined	159
src/optimizer/direct/ ParameterNormalizer.h	
In this file, the header of the ParameterNormalizer class is defined	160
src/optimizer/direct/ StoppingCondition.cpp	
In this file, the implementation of the StoppingCondition class is defined	161
src/optimizer/direct/ StoppingCondition.h	
In this file, the header of the StoppingCondition class is defined	162
src/optimizer/direct/hyrect/ BaseRect.cpp	
In this file, the implementation of the BaseRect class is defined	152
src/optimizer/direct/hyrect/ BaseRect.h	
In this file, the header of the BaseRect class is defined	153
src/optimizer/direct/hyrect/ ChildRect.cpp	
In this file, the implementation of the ChildRect class is defined	154
src/optimizer/direct/hyrect/ ChildRect.h	
In this file, the header of the ChildRect class is defined	155
src/optimizer/direct/hyrect/ HyRect.cpp	
In this file, the implementation of the HyRect class is defined	155
src/optimizer/direct/hyrect/ HyRect.h	
In this file, the header of the HyRect class is defined	156
src/parameters/ ContinuousParameter.cpp	
In this file, the implementation of the ContinuousParameter class is defined	165
src/parameters/ ContinuousParameter.h	
In this file, the header of the ContinuousParameter class is defined	165
src/parameters/ DiscreteParameter.cpp	
In this file, the implementation of the DiscreteParameter class is defined	166
src/parameters/ DiscreteParameter.h	
In this file, the header of the DiscreteParameter class is defined	167
src/parameters/ Parameter.cpp	
In this file, the implementation of the Parameter class is defined	168
src/parameters/ Parameter.h	
In this file, the header of the Parameter class is defined	169
src/parameters/ ParameterDefinition.cpp	
In this file, the implementation of the ParameterDefinition class is defined	170
src/parameters/ ParameterDefinition.h	
In this file, the header of the ParameterDefinition class is defined	171
src/runner/ SimulationRunner.cpp	
In this file, the implementation of the SimulationRunner class is defined	175
src/runner/ SimulationRunner.h	
In this file, the header of the SimulationRunner class is defined	175
src/runner/plex/ ConfigEditor.cpp	
In this file, the implementation of the ConfigEditor class is defined	172
src/runner/plex/ ConfigEditor.h	
In this file, the header of the ConfigEditor class is defined	173
src/runner/plex/ PlexSimulationRunner.cpp	
In this file, the implementation of the PlexSimulationRunner class is defined	174
src/runner/plex/ PlexSimulationRunner.h	
In this file, the header of the PlexSimulationRunner class is defined	174

src/status/ Status.cpp	
In this file, the implementation of the Status class is defined	176
src/status/ Status.h	
In this file, the header of the Status class is defined	177
src/status/ StatusBar.cpp	
In this file, the implementation of the StatusBar class is defined	178
src/status/ StatusBar.h	
In this file, the header of the StatusBar class is defined	178
src/utls/ Abortable.cpp	
In this file, the implementation of the Abortable class is defined	181
src/utls/ Abortable.h	
In this file, the header of the Abortable class is defined	181
src/utls/ CommandLine.cpp	
In this file, the implementation of the CommandLine class is defined	182
src/utls/ CommandLine.h	
In this file, the header of the CommandLine class is defined	183
src/utls/ Multithreaded.h	
In this file, the header of the Multithreaded class is defined	183
src/utls/ Multithreaded.tpp	
In this file, the implementation of the Multithreaded class is defined	184
src/utls/ PythonScript.cpp	
In this file, the implementation of the PythonScript class is defined	185
src/utls/ PythonScript.h	
In this file, the header of the PythonScript class is defined	186
src/utls/ ThreadSafeQueue.h	
In this file, the header of the ThreadSafeQueue class is defined	187
src/utls/ ThreadSafeQueue.tpp	
In this file, the implementation of the ThreadSafeQueue class is defined	188

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

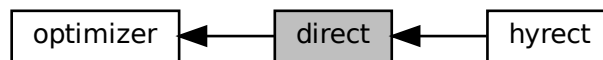
```
struct Controller::stepstate Controller::stepState [protected]
```

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 [StoppingCondition](#)

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

- class [ParameterNormalizer](#)

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

- class [Levels](#)

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

- class [GrahamScan](#)

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

Enumerations

- enum [level](#) : unsigned char {
`I2_0 = 0 , I1_1 = 1 , I0_2 = 2 , I1_3 = 3 ,
I1_4 = 4 , I0_5 = 5 , I1_6 = 6 , I2_7 = 7 }`

An enum representing the sequence of local levels.

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](#).

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 23 of file Levels.h.

7.3 plexe

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

Collaboration diagram for plexe:



Classes

- class [PlexeSimulationRunner](#)
A class capable of starting platooning simulations in the *Plexe* framework with given parameterCombinations.
- class [ConfigEditor](#)
A class capable of creating *.ini* files with certain options based on a complete *omnetpp.ini*.

7.3.1 Detailed Description

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

7.4 constant_headway

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

Collaboration diagram for constant_headway:



Files

- file [constant_headway.py](#)
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.

7.4.1 Detailed Description

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

7.5 parameters

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

Classes

- class [Parameter](#)
A class acting as the container of the value of a parameter defined by a [ParameterDefinition](#).
- class [ContinuousParameter](#)
Implements a [Parameter](#) using continuous values in the form of floating point numbers.
- class [ParameterDefinition](#)
A class storing information on the properties of parameters that are being optimized.
- class [DiscreteParameter](#)
Implements a [Parameter](#) using discrete values.

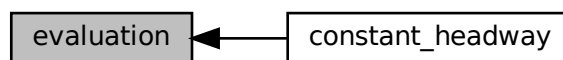
7.5.1 Detailed Description

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

7.6 evaluation

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

Collaboration diagram for evaluation:



Modules

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

Classes

- class [Evaluation](#)
A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization.

7.6.1 Detailed Description

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

Implementations must extend [Evaluation](#).

7.7 runner

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

Collaboration diagram for runner:



Modules

- [plexe](#)

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

Classes

- class [SimulationRunner](#)

A class capable of running simulations with certain parameterCombinations.

7.7.1 Detailed Description

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

Implementations must extend [SimulationRunner](#).

7.8 status

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

Classes

- class [Status](#)

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

- class [StatusBar](#)

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

Enumerations

- enum [step](#) : char { **INIT** = -1 , **OPTIMIZER** = 0 , **RUNNER** = 1 , **EVALUATION** = 2 }

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

7.8.1 Detailed Description

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

7.8.2 Enumeration Type Documentation

7.8.2.1 step

```
enum step : char
```

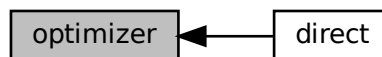
An Enum defining the steps, an optimization process cycles through.

Definition at line 29 of file StatusBar.h.

7.9 optimizer

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

Collaboration diagram for optimizer:



Modules

- [direct](#)

This module extends [Optimizer](#) to use a variant of the DIRECT algorithm by [Jones et al.](#)

Classes

- class [Optimizer](#)

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

7.9.1 Detailed Description

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

Implementations must extend [Optimizer](#).

7.10 hyrect

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

Collaboration diagram for hyrect:



Classes

- class [HyRect](#)
An abstract class representing a rectangular part of the search space.
- class [BaseRect](#)
A class representing a [HyRect](#) without a parent rectangle.
- class [ChildRect](#)
A class representing a [HyRect](#) that has a parent [HyRect](#).

Enumerations

- enum class [position](#) : char { **LEFT** = 0 , **MIDDLE** = 1 , **RIGHT** = 2 , **BASE** = -1 }
An enum representing the position of a [HyRect](#) relative to its parent [HyRect](#).

7.10.1 Detailed Description

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

7.10.2 Enumeration Type Documentation

7.10.2.1 position

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

An enum representing the position of a [HyRect](#) relative to its parent [HyRect](#).

If it is a [BaseRect](#) and therefore has no parent, BASE is used.

Definition at line 35 of file HyRect.h.

7.11 utils

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

Files

- file [Types.h](#)
In this file, types are defined which should be used across the whole framework.
- file [ComparisonFunctions.h](#)
In this file, comparison functions are defined which should be used across the whole framework.
- file [main.cpp](#)
In this file, the main function running the Simopticon framework is defined.
- file [main.cpp](#)
In this file, the main function running the Simopticon framework is defined.

Classes

- class [PythonScript](#)
A class containing functionality for interfacing with the function of a Python module on creation.
- class [Abortable](#)
A simple interface for classes that encapsulate abortable processes.
- class [Multithreaded](#)< [Key](#), [T](#), [Compare](#), [Allocator](#) >
A class implementing concurrent execution of the same function for different arguments.
- class [ThreadsafeQueue](#)< [Key](#) >
A container class of a queue that is safe for concurrent access of different threads.
- class [CommandLine](#)
A class containing functionality for executing commands on UNIX shell.

7.11.1 Detailed Description

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

Chapter 8

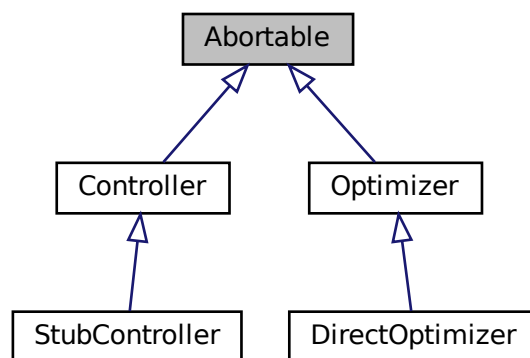
Class Documentation

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.
Definition at line 13 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 8 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 18 of file `Abortable.h`.

Referenced by `abort()`.

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

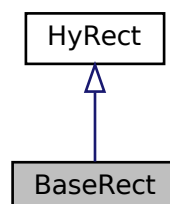
- `src/utils/Abortable.h`
- `src/utils/Abortable.cpp`

8.2 BaseRect Class Reference

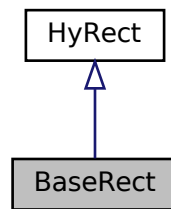
A class representing a [HyRect](#) without a parent rectangle.

```
#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.

Definition at line 16 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

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

Definition at line 8 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 11 of file `BaseRect.cpp`.

References `HyRect::D`.

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

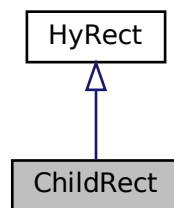
- `src/optimizer/direct/hyrect/BaseRect.h`
- `src/optimizer/direct/hyrect/BaseRect.cpp`

8.3 ChildRect Class Reference

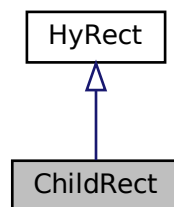
A class representing a [HyRect](#) that has a parent [HyRect](#).

```
#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 $t > 0$.

Definition at line 16 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 10 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 15 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 33 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 21 of file [ChildRect.h](#).

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

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

- [src/optimizer/direct/hyrect/ChildRect.h](#)
- [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](#):

[CmpPairVectorSharedParameterFunctionvalue](#)

Public Member Functions

- bool [operator\(\)](#) (const std::pair< [parameterCombination](#), [functionValue](#) > &a, const std::pair< [parameterCombination](#), [functionValue](#) > &b) const

Compares two pairs of [parameterCombination](#) and [functionValue](#).

8.4.1 Detailed Description

This struct implements the comparison of two pairs of parameterCombination and function value.
Definition at line 55 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 62 of file ComparisonFunctions.h.

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

- 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:



```
graph TD
    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 42 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 49 of file ComparisonFunctions.h.

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

- [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 17 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 24 of file `DirectComparisonFunctions.h`.

References [HyRect::getAvgValue\(\)](#), and [HyRect::getSamplingVertices\(\)](#).

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

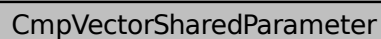
- `src/optimizer/direct/DirectComparisonFunctions.h`

8.7 CmpVectorSharedParameter Struct Reference

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

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

Collaboration diagram for `CmpVectorSharedParameter`:



```

classDiagram
    class 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 19 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 26 of file ComparisonFunctions.h.

References `Parameter::operator!=()`, and `Parameter::operator<()`.

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

- [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:



```

classDiagram
    class 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.

Definition at line 26 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 12 of file CommandLine.cpp.

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

- [src/utis/CommandLine.h](#)
- [src/utis/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`.
Definition at line 24 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

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

Definition at line 11 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

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

Definition at line 16 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 95 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 85 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 74 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 99 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 90 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 47 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 .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 61 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 65 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 34 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 43 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 30 of file `ConfigEditor.h`.

8.9.4.4 RESULTS

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

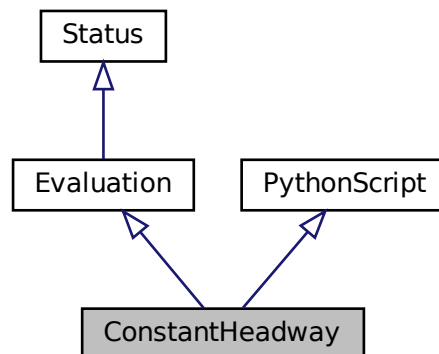
Path to the `optResults` directory in [DIR](#) where the simulation result files are generated.
Definition at line 38 of file `ConfigEditor.h`.
The documentation for this class was generated from the following files:

- `src/runner/plexe/ConfigEditor.h`
- `src/runner/plexe/ConfigEditor.cpp`

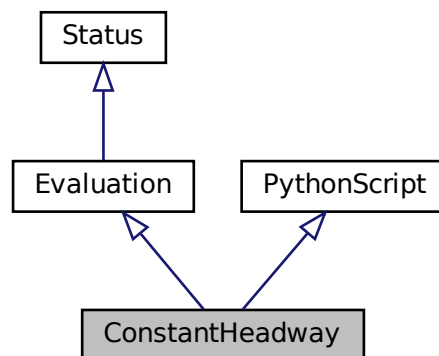
8.10 ConstantHeadway Class Reference

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 35 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 10 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 75 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 79 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 83 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 > ConstantHeadway::processOutput (
```

```
    const std::set< std::pair< std::filesystem::path, std::set< runId >>> & experimentResults ) [override], [virtual]
```

Returns values to the results of multiple simulation runs.

Passes given parameters to the multithreaded function of [constant_headway.py](#).

Parameters

<i>experimentResults</i>	Paths to and identifiers of the simulation results.
--------------------------	---

Returns

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

Reimplemented from [Evaluation](#).

Definition at line 15 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 60 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 65 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 41 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 46 of file ConstantHeadway.h.

Referenced by [getStatusBar\(\)](#), and [processOutput\(\)](#).

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

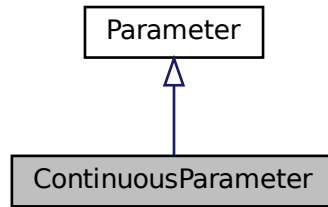
- [src/evaluation/constant_headway/ConstantHeadway.h](#)
- [src/evaluation/constant_headway/ConstantHeadway.cpp](#)

8.11 ContinuousParameter Class Reference

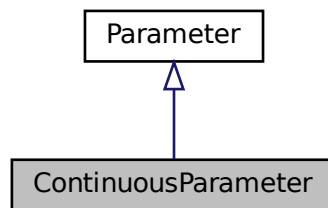
Implements a [Parameter](#) using continuous values in the form of floating point numbers.

```
#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.
Definition at line 15 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 11 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 18 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 22 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

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

Implements [Parameter](#).

Definition at line 26 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 20 of file `ContinuousParameter.h`.

Referenced by `ContinuousParameter()`, `getVal()`, and `setVal()`.

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

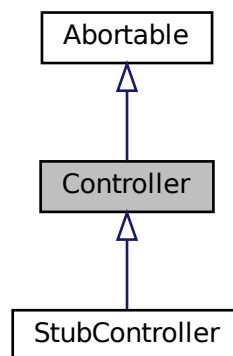
- `src/parameters/ContinuousParameter.h`
- `src/parameters/ContinuousParameter.cpp`

8.12 Controller Class Reference

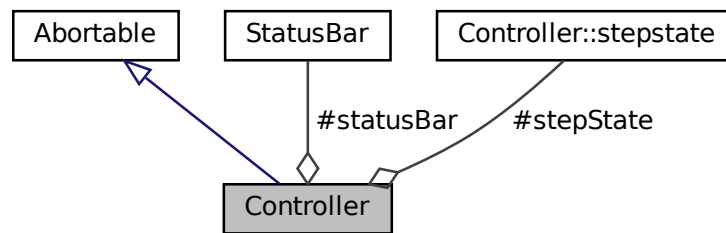
A class responsible for communication between [Optimizer](#), [SimulationRunner](#) and [Evaluation](#) and also user interaction such as tracking results, updating [StatusBar](#) and handling interrupts by the user via [Abortable](#).

```
#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](#).
Definition at line 43 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

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

Definition at line 37 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 265 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 198 of file `Controller.cpp`.

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

8.12.3.3 getValueMap()

`ValueMap & Controller::getValueMap ()`

Returns [valueMap](#).

Returns

A [ValueMap](#) object.

Definition at line 163 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 214 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 129 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 167 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 > >, CmpVectorSharedParameter > > 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

<i>runs</i>	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 191 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 232 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 257 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 113 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 48 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 105 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 53 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 109 of file `Controller.h`.

Referenced by `updateStatus()`.

8.12.4.6 statusBar

`StatusBar` Controller::statusBar [protected]

`StatusBar` object used for output.

Definition at line 101 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 61 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 57 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 117 of file Controller.h.

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

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

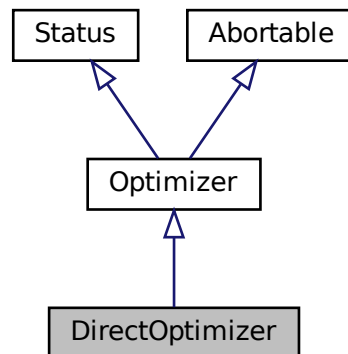
- `src/controller/Controller.h`
- `src/controller/Controller.cpp`

8.13 DirectOptimizer Class Reference

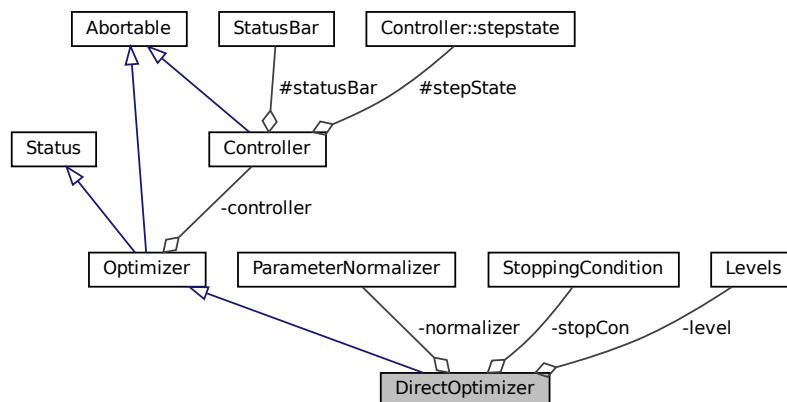
A class capable of finding the minimum of a blackbox function using the DIRECT algorithm.

```
#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, StoppingCondition 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.
- `StoppingCondition stopCon`
An object deciding when the optimization stops.
- `Levels level`
An object used switching between different levels between global and local search.
- `ParameterNormalizer normalizer`
An object used for transformation between the unit hypercube used in DIRECT and the actual parameter space.
- `bool trackProgress`
Defines if the current number of iterations, evaluations, rectangles and the optimal value should be recorded into a `.csv` file after each iteration.
- `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 39 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,
    StoppingCondition 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>	StoppingCondition defining the end of optimization.
<i>trackProgress</i>	Defines whether the progress should be printed in a <code>.csv</code> file.

Definition at line 16 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 111 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 91 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 169 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 186 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 173 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 182 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 73 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 95 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 146 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 24 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 157 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<> > DirectOptimizer::activeRects [private]
```

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

This includes all rectangles which have not been divided yet. They are grouped by [HyRect::t](#) and sorted by [HyRect::avgValue](#) which simplifies the search for potentially optimal rectangles in [optimalRectangles](#).

Definition at line 72 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 44 of file `DirectOptimizer.h`.

8.13.4.3 iterations

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

Number of iterations completed.

Definition at line 48 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 56 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 60 of file `DirectOptimizer.h`.

8.13.4.6 stopCon

`StoppingCondition DirectOptimizer::stopCon [private]`

An object deciding when the optimization stops.

Definition at line 52 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 65 of file DirectOptimizer.h.

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

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

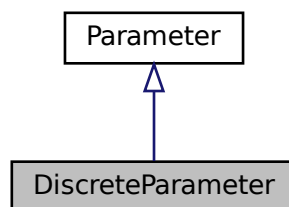
- `src/optimizer/direct/DirectOptimizer.h`
- `src/optimizer/direct/DirectOptimizer.cpp`

8.14 DiscreteParameter Class Reference

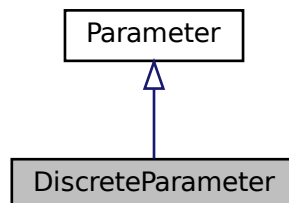
Implements a [Parameter](#) using discrete values.

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

Inheritance diagram for DiscreteParameter:



Collaboration diagram for DiscreteParameter:



Public Member Functions

- [DiscreteParameter](#) (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.

8.14.1 Detailed Description

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

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

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

8.14.2 Constructor & Destructor Documentation

8.14.2.1 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 12 of file [DiscreteParameter.cpp](#).

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

8.14.2.2 `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

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

Definition at line 22 of file `DiscreteParameter.cpp`.
 References `DiscreteParameter()`, `Parameter::getMax()`, and `Parameter::getMin()`.

8.14.3 Member Function Documentation

8.14.3.1 `getOffset()`

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

Returns the value of `offset`.

Returns

A floating point number representing the offset.

Definition at line 42 of file `DiscreteParameter.cpp`.
 References `offset`.

8.14.3.2 `getStep()`

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

Returns the value of `step`.

Returns

A floating point number representing the difference between discrete values.

Definition at line 38 of file `DiscreteParameter.cpp`.
 References `step`.

8.14.3.3 `getTimes()`

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

Returns the value of `times`.

Returns

An integer representing the times value.

Definition at line 26 of file `DiscreteParameter.cpp`.
 References `times`.

8.14.3.4 getVal()

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

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

Returns

A coordinate representing the value of the [ContinuousParameter](#).

Implements [Parameter](#).

Definition at line 46 of file `DiscreteParameter.cpp`.

References `offset`, `step`, and `times`.

8.14.3.5 setTimes()

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

Sets the value of `times` to the given value.

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

Parameters

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

Definition at line 30 of file `DiscreteParameter.cpp`.

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

Referenced by `setVal()`.

8.14.3.6 setVal()

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

Sets the value of the [DiscreteParameter](#) to the discrete value closest to the given value by modifying `times` using `setTimes`.

Parameters

<i>val</i>	Value to set the DiscreteParameter to.
------------	--

Implements [Parameter](#).

Definition at line 50 of file `DiscreteParameter.cpp`.

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

8.14.4 Member Data Documentation

8.14.4.1 offset

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

Offset used in the value calculation.

Definition at line 29 of file `DiscreteParameter.h`.

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

8.14.4.2 step

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

Difference between discrete values.

Used in the value calculation.

Definition at line 25 of file DiscreteParameter.h.

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

8.14.4.3 times

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

Times used in the value calculation.

Definition at line 21 of file DiscreteParameter.h.

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

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

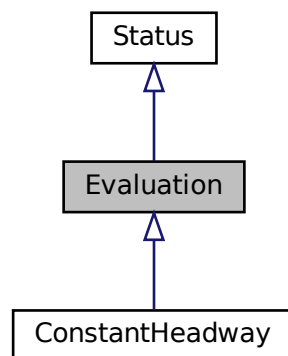
- [src/parameters/DiscreteParameter.h](#)
- [src/parameters/DiscreteParameter.cpp](#)

8.15 Evaluation Class Reference

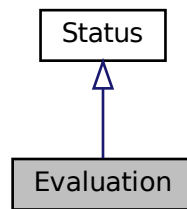
A class capable of evaluating simulation results and scoring them with a value which is treated as the function value for the optimization.

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

Inheritance diagram for Evaluation:



Collaboration diagram for Evaluation:



Public Member Functions

- virtual `functionValue processOutput` (`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

8.15.1 Detailed Description

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

A lower value is considered better in this framework. The optimized function can be viewed as an error function.

Definition at line 32 of file `Evaluation.h`.

8.15.2 Member Function Documentation

8.15.2.1 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 17 of file `Evaluation.cpp`.

References `Status::getName()`.

8.15.2.2 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 21 of file Evaluation.cpp.

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

8.15.2.3 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 25 of file Evaluation.cpp.

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

8.15.2.4 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 9 of file Evaluation.cpp.

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

8.15.2.5 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:

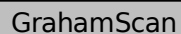
- `src/evaluation/Evaluation.h`
- `src/evaluation/Evaluation.cpp`

8.16 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:



```

classDiagram
    class 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.

8.16.1 Detailed Description

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

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

8.16.2 Member Function Documentation

8.16.2.1 `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 12 of file `GrahamScan.cpp`.

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

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

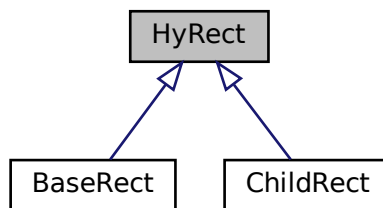
- `src/optimizer/direct/GrahamScan.h`
- `src/optimizer/direct/GrahamScan.cpp`

8.17 HyRect Class Reference

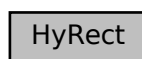
An abstract class representing a rectangular part of the search space.

```
#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`.

8.17.1 Detailed Description

An abstract class representing a rectangular part of the search space.
Definition at line 43 of file HyRect.h.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 HyRect()

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

Creates a [HyRect](#) with the given dimensionality, position and depth.

Parameters

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

Definition at line 11 of file HyRect.cpp.

References *D*, *pos*, and *t*.

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

8.17.3 Member Function Documentation

8.17.3.1 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 14 of file HyRect.cpp.

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

8.17.3.2 getAvgValue()

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

Returns the value of [avgValue](#).

Returns

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

Definition at line 40 of file HyRect.cpp.

References [avgValue](#).

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

8.17.3.3 getD()

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

Returns the value of [D](#).

Returns

A dimension representing the number of dimensions of the rectangle.

Definition at line 72 of file HyRect.cpp.

References [D](#).

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

8.17.3.4 getDepth()

```
depth HyRect::getDepth ( ) const
```

Returns the value of [t](#).

Returns

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

Definition at line 36 of file HyRect.cpp.

References [t](#).

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

8.17.3.5 getDiagonalLength()

```
dirCoordinate HyRect::getDiagonalLength ( ) const
```

Returns the length of the diagonal of the rectangle.

Basically calculates the euclidian distance between the vertices returned by [getSamplingVertices](#). Instead of actually invoking the costly recursive [getSamplingVertices](#) function, a calculation based on [t](#) is executed

Returns

A dirCoordinate representing the diagonal length of the rectangle.

Definition at line 21 of file HyRect.cpp.

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

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

8.17.3.6 getPos()

```
position HyRect::getPos ( ) const
```

Returns the value of [pos](#).

Returns

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

Definition at line 32 of file HyRect.cpp.

References [pos](#).

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

8.17.3.7 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>\(\)\(\)](#).

8.17.3.8 getSplitDim()

```
dimension HyRect::getSplitDim ( ) const
```

Calculates the dimension where this rectangle must be or has been split by [divide](#).

Since the split dimensions are simply chosen in ascending order the calculations only needs the depth stored in [t](#).

Returns

A dimension where the [HyRect](#) has been oder will be split.

Definition at line 28 of file HyRect.cpp.

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

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

8.17.3.9 operator"!="()

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

Checks if the current and the given [HyRect](#) objects are unequal by comparing their [pos](#), [D](#), and [t](#).

Basically negates [operator==](#).

Parameters

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

Returns

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

Definition at line 56 of file HyRect.cpp.

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

8.17.3.10 operator<()

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

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

Parameters

<i>rect</i>	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 52 of file HyRect.cpp.

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

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

8.17.3.11 operator<=()

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

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

Basically negates [operator>](#).

Parameters

<i>rhs</i>	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 64 of file HyRect.cpp.

References operator<().

8.17.3.12 operator==()

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

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

Parameters

<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 48 of file HyRect.cpp.

References *D*, *pos*, and *t*.

Referenced by operator!=().

8.17.3.13 operator>()

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

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

Basically calls operator< on the switched inputs.

Parameters

<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 60 of file HyRect.cpp.

References operator<().

8.17.3.14 operator>=()

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

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

Basically negates operator<.

Parameters

<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 68 of file `HyRect.cpp`.

References `operator<()`.

8.17.3.15 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 44 of file `HyRect.cpp`.

References `avgValue`.

8.17.4 Member Data Documentation

8.17.4.1 avgValue

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

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

Definition at line 63 of file `HyRect.h`.

Referenced by `getAvgValue()`, `operator<()`, and `setAvgValue()`.

8.17.4.2 D

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

Dimensionality of the rectangle.

Is equivalent to the dimensionality of the search space, i.e. the number of optimized parameters.

Definition at line 49 of file `HyRect.h`.

Referenced by `HyRect()`, `getD()`, `getDiagonalLength()`, `BaseRect::getSamplingVertices()`, `getSplitDim()`, and `operator==()`.

8.17.4.3 pos

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

Position of the rectangle relative to its parent rectangle.

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

Definition at line 59 of file `HyRect.h`.

Referenced by `HyRect()`, `getPos()`, `ChildRect::getSamplingVertices()`, `operator==()`, and `ChildRect::operator==()`.

8.17.4.4 `t`

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

Depth of the rectangle in the partition tree.

Equal to the number of transitive parent rectangles. For `BaseRect`, `t` is always 0.

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

Referenced by `HyRect()`, `getDepth()`, `getDiagonalLength()`, `getSplitDim()`, `operator<()`, and `operator==()`.

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

- `src/optimizer/direct/hyrect/HyRect.h`
- `src/optimizer/direct/hyrect/HyRect.cpp`

8.18 Levels Class Reference

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

`#include "Levels.h"`

Collaboration diagram for Levels:



```

classDiagram
    class 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.

8.18.1 Detailed Description

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

Definition at line 31 of file Levels.h.

8.18.2 Member Function Documentation

8.18.2.1 [getEpsilon\(\)](#)

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

Returns the epsilon value on the current level the DIRECT algorithm resides on.

Either [L3_EPSILON](#), [L2_EPSILON](#), [L1_EPSILON](#) or [L0_EPSILON](#).

Returns

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

Definition at line 58 of file Levels.cpp.

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

8.18.2.2 [getLevel\(\)](#)

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

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

Returns

An integral corresponding to the current level.

Definition at line 71 of file Levels.cpp.

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

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

8.18.2.3 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 20 of file Levels.cpp.

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

8.18.2.4 isGlobal()

```
bool Levels::isGlobal ( ) const
```

Returns the value of [global](#).

Returns

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

Definition at line 87 of file Levels.cpp.

References [global](#).

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

8.18.2.5 nextLevel()

```
unsigned char Levels::nextLevel ( )
```

Switches [currentLevel](#) to the next local level if [global](#) is false.

Returns

A number representing the current level after switching.

Definition at line 12 of file Levels.cpp.

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

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

8.18.2.6 setGlobal()

```
void Levels::setGlobal (
    bool val )
```

Sets the value of [global](#).

Parameters

<i>val</i>	Defines whether global optimization should be used in the following iterations.
------------	---

Definition at line 91 of file Levels.cpp.

References global.

Referenced by DirectOptimizer::runOptimization().

8.18.3 Member Data Documentation

8.18.3.1 currentLevel

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

Local level the optimization is currently using when `global` is *false*.

Definition at line 36 of file Levels.h.

Referenced by `getLevel()`, and `nextLevel()`.

8.18.3.2 global

```
bool Levels::global = false [private]
```

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

Definition at line 40 of file Levels.h.

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

8.18.3.3 L0_EPSILON

```
constexpr static const double Levels::L0_EPSILON = 0 [static], [constexpr]
```

Epsilon value to be used when DIRECT algorithm uses level 0.

Definition at line 58 of file Levels.h.

Referenced by `getEpsilon()`.

8.18.3.4 L0_SIZE

```
constexpr static const long double Levels::L0_SIZE = 0.04 [static], [constexpr]
```

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

Definition at line 75 of file Levels.h.

Referenced by `getRectSubset()`.

8.18.3.5 L1_EPSILON

```
constexpr static const double Levels::L1_EPSILON = 1e-7 [static], [constexpr]
```

Epsilon value to be used when DIRECT algorithm uses level 1.

Definition at line 54 of file Levels.h.

Referenced by `getEpsilon()`.

8.18.3.6 L1_SIZE

```
constexpr static const long double Levels::L1_SIZE = 0.95 [static], [constexpr]
```

Fraction of rectangles in partition to be used on level 1 (only smaller rectangles are considered).

Definition at line 71 of file Levels.h.

Referenced by `getRectSubset()`.

8.18.3.7 L2_EPSILON

```
constexpr static const double Levels::L2_EPSILON = 1e-5 [static], [constexpr]
```

Epsilon value to be used when DIRECT algorithm uses level 2.

Definition at line 50 of file Levels.h.
Referenced by getEpsilon().

8.18.3.8 L2_SIZE

```
constexpr static const long double Levels::L2_SIZE = 1 [static], [constexpr]
```

Fraction of rectangles in partition to be used on level 2 (only smaller rectangles are considered).
Definition at line 67 of file Levels.h.
Referenced by getRectSubset().

8.18.3.9 L3_EPSILON

```
constexpr static const double Levels::L3_EPSILON = 1e-5 [static], [constexpr]
```

Epsilon value to be used when DIRECT algorithm uses level 3.
Definition at line 46 of file Levels.h.
Referenced by getEpsilon(), and DirectOptimizer::runOptimization().

8.18.3.10 L3_SIZE

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

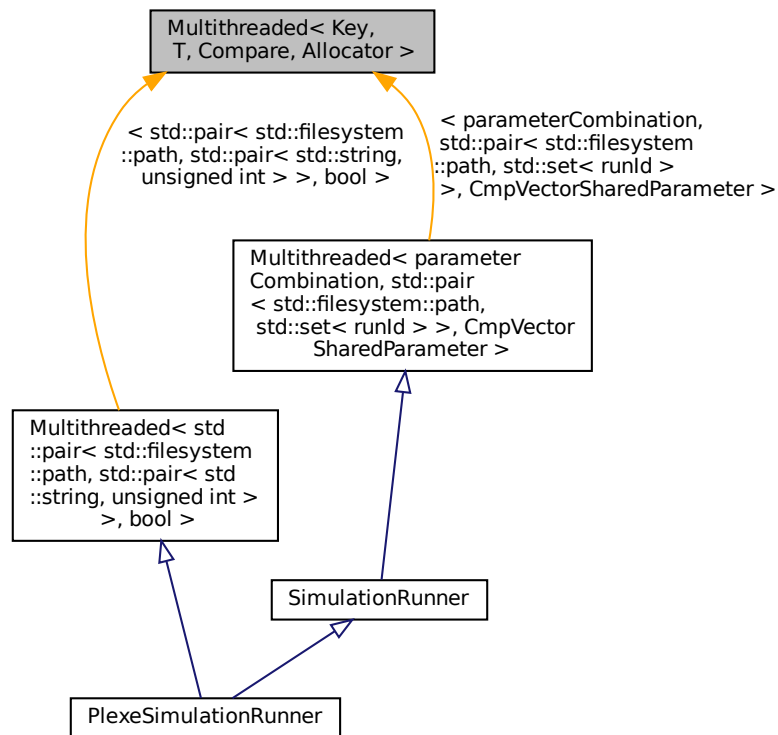
Fraction of rectangles in partition to be used on level 3 (only larger rectangles are considered).
Definition at line 63 of file Levels.h.
Referenced by getRectSubset().
The documentation for this class was generated from the following files:

- [src/optimizer/direct/Levels.h](#)
- [src/optimizer/direct/Levels.cpp](#)

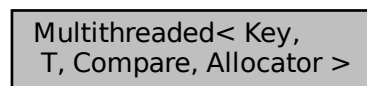
8.19 Multithreaded< Key, T, Compare, Allocator > Class Template Reference

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.

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

Definition at line 26 of file [Multithreaded.h](#).

8.19.2 Constructor & Destructor Documentation

8.19.2.1 Multithreaded()

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

Creates a [Multithreaded](#) class that does not use more than the given number of threads.

Parameters

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

Definition at line 11 of file [Multithreaded.tpp](#).

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

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

8.19.3 Member Function Documentation

8.19.3.1 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 35 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()`.

8.19.3.2 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 16 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()`.

8.19.3.3 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()`.

8.19.4 Member Data Documentation

8.19.4.1 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 39 of file Multithreaded.h.

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

8.19.4.2 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 43 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:

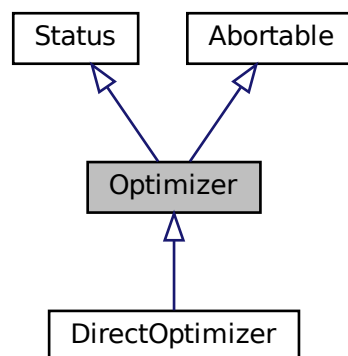
- `src/utils/Multithreaded.h`
- `src/utils/Multithreaded.tpp`

8.20 Optimizer Class Reference

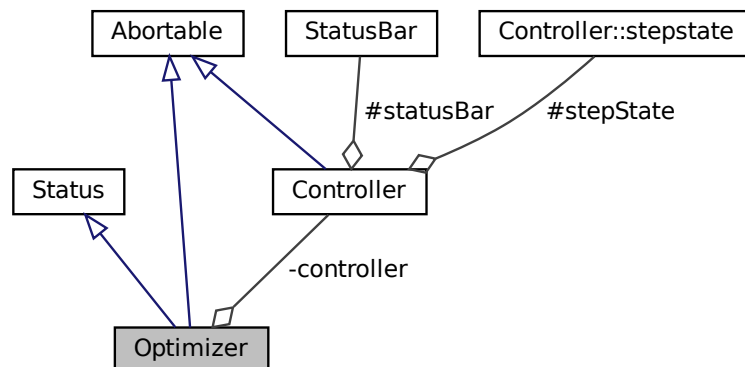
A class containing an optimization strategy which searches the minimum of a blackbox function given through argument-value pairs.

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

Inheritance diagram for Optimizer:



Collaboration diagram for `Optimizer`:



Public Member Functions

- `Optimizer (Controller &ctrl, 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

8.20.1 Detailed Description

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

The [Optimizer](#) has control over which 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.

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

8.20.2 Constructor & Destructor Documentation

8.20.2.1 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 17 of file `Optimizer.cpp`.

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

8.20.3 Member Function Documentation

8.20.3.1 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 25 of file `Optimizer.cpp`.

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

8.20.3.2 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 29 of file `Optimizer.cpp`.

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

8.20.3.3 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 33 of file Optimizer.cpp.

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

8.20.3.4 getValueMap()

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

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

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

Returns

Definition at line 21 of file Optimizer.cpp.

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

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

8.20.3.5 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 13 of file Optimizer.cpp.

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

8.20.3.6 runOptimization()

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

Starts the optimization process.

Should only return if the optimization strategy deems the optimization complete or when [abort](#) is called.

Implemented in [DirectOptimizer](#).

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

8.20.4 Member Data Documentation

8.20.4.1 controller

`Controller& Optimizer::controller [private]`

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

Definition at line 47 of file `Optimizer.h`.

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

8.20.4.2 parameters

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

List of parameters to be optimized.

Definition at line 53 of file `Optimizer.h`.

Referenced by `Optimizer()`.

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

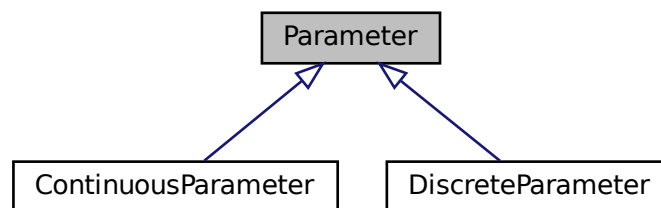
- `src/optimizer/Optimizer.h`
- `src/optimizer/Optimizer.cpp`

8.21 Parameter Class Reference

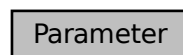
A class acting as the container of the value of a parameter defined by a [ParameterDefinition](#).

`#include "Parameter.h"`

Inheritance diagram for `Parameter`:



Collaboration diagram for `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](#).

8.21.1 Detailed Description

A class acting as the container of the value of a parameter defined by a [ParameterDefinition](#).
Definition at line 30 of file [Parameter.h](#).

8.21.2 Constructor & Destructor Documentation

8.21.2.1 [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 12 of file [Parameter.cpp](#).

References [definition](#).

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

8.21.3 Member Function Documentation

8.21.3.1 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 28 of file `Parameter.cpp`.

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

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

8.21.3.2 getMax()

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

Returns the maximum value of the [Parameter](#) stored in [ParameterDefinition::max](#) of [definition](#).

Returns

A coordinate representing the maximum value.

Definition at line 20 of file `Parameter.cpp`.

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

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

8.21.3.3 getMin()

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

Returns the minimum value of the [Parameter](#) stored in [ParameterDefinition::min](#) of [definition](#).

Returns

A coordinate representing the minimum value.

Definition at line 16 of file `Parameter.cpp`.

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

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

8.21.3.4 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 24 of file `Parameter.cpp`.

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

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

8.21.3.5 getVal()

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

Returns the current value of the [Parameter](#).

Returns

A coordinate representing the value of the [Parameter](#).

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

Referenced by [ValueMap::isTopValue\(\)](#), [ParameterNormalizer::normalize\(\)](#), [operator<\(\)](#), [operator==\(\)](#), and [StatusBar::printResult\(\)](#).

8.21.3.6 operator!=()

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

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

Parameters

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

Returns

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

Definition at line 36 of file [Parameter.cpp](#).

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

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

8.21.3.7 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 40 of file [Parameter.cpp](#).

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

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

8.21.3.8 operator<=()

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

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

Basically negates [operator<](#).

Parameters

<i>rhs</i>	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 48 of file `Parameter.cpp`.

References `operator<()`.

8.21.3.9 operator==()

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

Checks if the current and the given [Parameter](#) objects are equal by comparing their value and [definition](#).

Parameters

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

Returns

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

Definition at line 32 of file `Parameter.cpp`.

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

Referenced by `operator!=()`.

8.21.3.10 operator>()

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

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

Basically calls `operator<` on the switched inputs.

Parameters

<i>rhs</i>	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 44 of file `Parameter.cpp`.

References `operator<()`.

8.21.3.11 operator>=()

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

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

Basically negates `operator>`.

Parameters

<i>rhs</i>	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 52 of file `Parameter.cpp`.

References `operator<()`.

8.21.3.12 setVal()

```
virtual void Parameter::setVal (
    coordinate val ) [pure virtual]
```

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

Parameters

<i>val</i>	Value to set the Parameter to.
------------	--

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

8.21.4 Member Data Documentation**8.21.4.1 definition**

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

Reference to the defining [ParameterDefinition](#).

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

Referenced by `Parameter()`, `getConfig()`, `getMax()`, `getMin()`, `getUnit()`, and `operator==()`.

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

- `src/parameters/Parameter.h`
- `src/parameters/Parameter.cpp`

8.22 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).

8.22.1 Detailed Description

A class storing information on the properties of parameters that are being optimized.
Definition at line 17 of file ParameterDefinition.h.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 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 11 of file ParameterDefinition.cpp.
References [config](#), [max](#), [min](#), and [unit](#).

8.22.3 Member Function Documentation

8.22.3.1 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 35 of file ParameterDefinition.cpp.

References config.

Referenced by Parameter::getConfig().

8.22.3.2 getMax()

`coordinate` ParameterDefinition::getMax () const

Returns the maximum value of the [Parameter](#) stored in [max](#).

Returns

A coordinate representing the maximum value.

Definition at line 27 of file ParameterDefinition.cpp.

References max.

Referenced by ParameterNormalizer::denormalize(), and Parameter::getMax().

8.22.3.3 getMin()

`coordinate` ParameterDefinition::getMin () const

Returns the minimum value of the [Parameter](#) stored in [min](#).

Returns

A coordinate representing the minimum value.

Definition at line 23 of file ParameterDefinition.cpp.

References min.

Referenced by ParameterNormalizer::denormalize(), and Parameter::getMin().

8.22.3.4 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 31 of file ParameterDefinition.cpp.

References unit.

Referenced by Parameter::getUnit().

8.22.4 Member Data Documentation

8.22.4.1 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 35 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getConfig().

8.22.4.2 max

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

Maximum value of the [Parameter](#).

Definition at line 26 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getMax().

8.22.4.3 min

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

Minimum value of the [Parameter](#).

Definition at line 22 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getMin().

8.22.4.4 unit

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

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

Definition at line 30 of file ParameterDefinition.h.

Referenced by ParameterDefinition(), and getUnit().

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

- src/parameters/[ParameterDefinition.h](#)
- src/parameters/[ParameterDefinition.cpp](#)

8.23 ParameterNormalizer Class Reference

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

```
#include "ParameterNormalizer.h"
```

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.

8.23.1 Detailed Description

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

Definition at line 24 of file `ParameterNormalizer.h`.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 ParameterNormalizer()

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

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

Parameters

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

Definition at line 13 of file `ParameterNormalizer.cpp`.

References `parameters`.

8.23.3 Member Function Documentation

8.23.3.1 denormalize()

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

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

Parameters

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

Returns

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

Definition at line 25 of file `ParameterNormalizer.cpp`.

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

8.23.3.2 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 17 of file ParameterNormalizer.cpp.

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

8.23.4 Member Data Documentation

8.23.4.1 parameters

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

[ParameterDefinition](#) of the optimized parameters.

Definition at line 29 of file ParameterNormalizer.h.

Referenced by `ParameterNormalizer()`, and `denormalize()`.

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

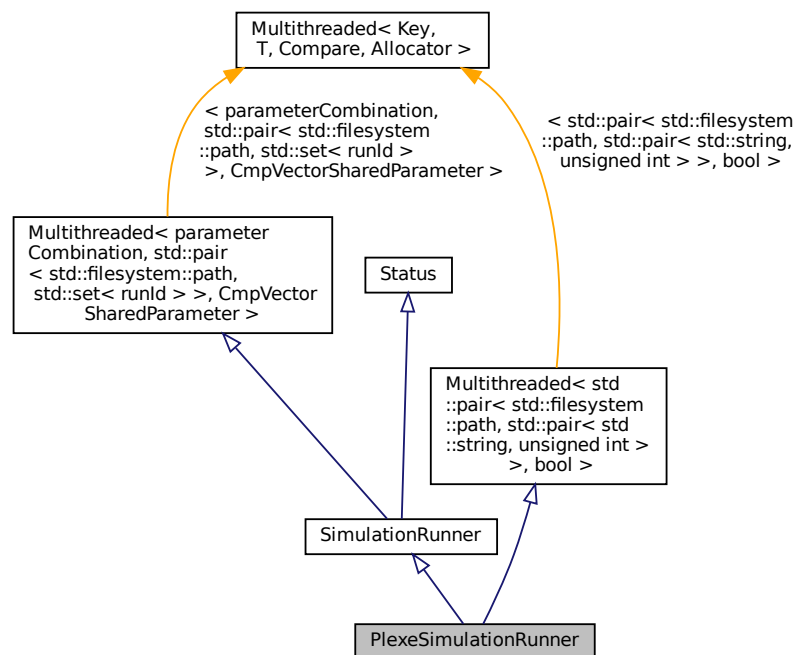
- `src/optimizer/direct/ParameterNormalizer.h`
- `src/optimizer/direct/ParameterNormalizer.cpp`

8.24 PlexeSimulationRunner Class Reference

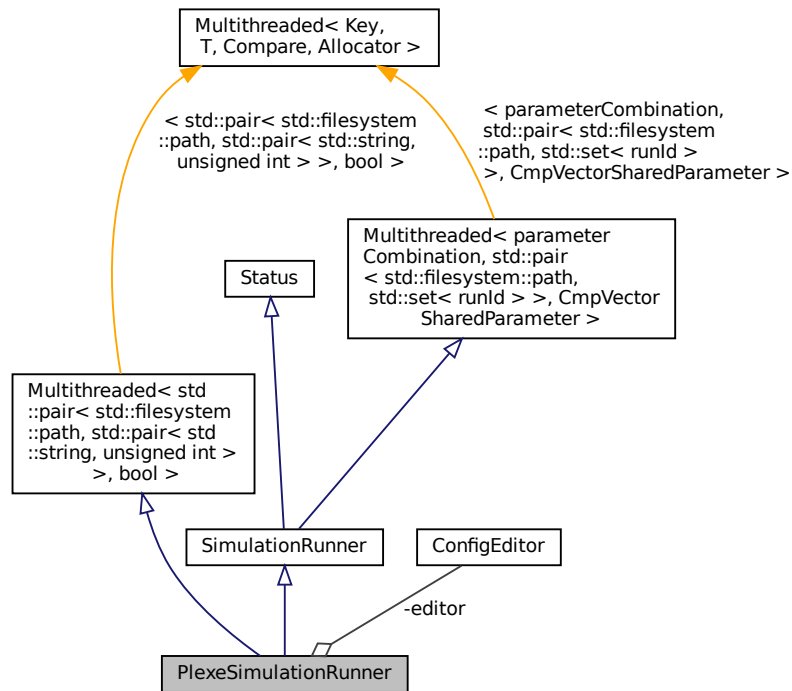
A class capable of starting platooning simulations in the [Plexe](#) framework with given parameterCombinations.

`#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

8.24.1 Detailed Description

A class capable of starting platooning simulations in the [Plexe](#) framework with given parameterCombinations. Definition at line 30 of file PlexeSimulationRunner.h.

8.24.2 Constructor & Destructor Documentation

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

Parameters

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

Definition at line 14 of file PlexeSimulationRunner.cpp.

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

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

8.24.3 Member Function Documentation

8.24.3.1 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 82 of file PlexeSimulationRunner.cpp.

8.24.3.2 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 58 of file PlexeSimulationRunner.cpp.

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

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

8.24.3.3 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 86 of file PlexeSimulationRunner.cpp.

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

8.24.3.4 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 99 of file PlexeSimulationRunner.cpp.

8.24.3.5 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 23 of file PlexeSimulationRunner.cpp.

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

8.24.3.6 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 67 of file PlexeSimulationRunner.cpp.

8.24.4 Member Data Documentation

8.24.4.1 editor

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

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

Definition at line 47 of file PlexeSimulationRunner.h.

8.24.4.2 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 37 of file PlexeSimulationRunner.h.

Referenced by [PlexeSimulationRunner\(\)](#), and [getStatus\(\)](#).

8.24.4.3 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 53 of file PlexeSimulationRunner.h.

Referenced by [getRunId\(\)](#), and [getStatus\(\)](#).

8.24.4.4 runNumberLock

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

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

Definition at line 57 of file PlexeSimulationRunner.h.

Referenced by `getRunId()`.

8.24.4.5 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 42 of file PlexeSimulationRunner.h.

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

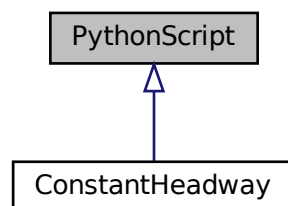
- `src/runner/plexe/`[PlexeSimulationRunner.h](#)
- `src/runner/plexe/`[PlexeSimulationRunner.cpp](#)

8.25 PythonScript Class Reference

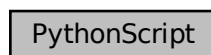
A class containing functionality for interfacing with the function of a Python module on creation.

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

Inheritance diagram for PythonScript:



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.

8.25.1 Detailed Description

A class containing functionality for interfacing with the function of a Python module on creation.
 See <https://docs.python.org/3/c-api/index.html> for more information.
 Definition at line 23 of file PythonScript.h.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 PythonScript()

```
PythonScript::PythonScript (
    const std::filesystem::path & path,
    const char * functionName )
```

Creates a connection to the given function of a Python script at the given path.

Parameters

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

Definition at line 11 of file PythonScript.cpp.
 Referenced by ConstantHeadway::ConstantHeadway().

8.25.2.2 ~PythonScript()

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

Ends connection to function [pFunc](#) and module [pModule](#).
 Definition at line 35 of file PythonScript.cpp.

8.25.3 Member Data Documentation

8.25.3.1 pFunc

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

Pointer to function which should be used by the class.
 Definition at line 32 of file PythonScript.h.

8.25.3.2 pModule

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

Pointer to module that contains function which should be used by the class.
 Definition at line 28 of file PythonScript.h.
 The documentation for this class was generated from the following files:

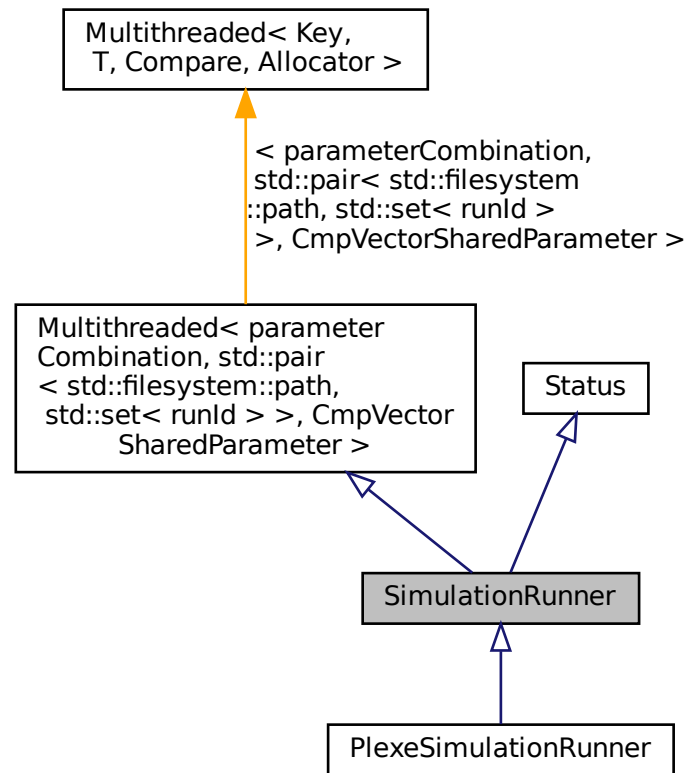
- src/utis/[PythonScript.h](#)
- src/utis/[PythonScript.cpp](#)

8.26 SimulationRunner Class Reference

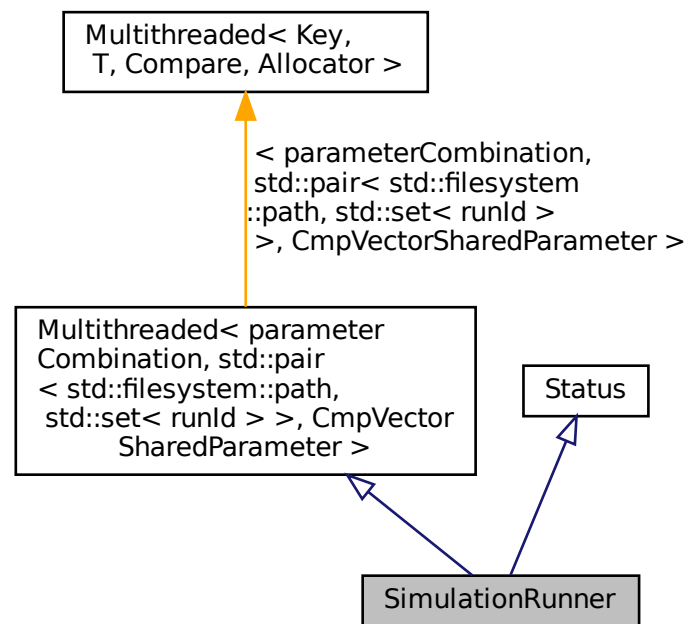
A class capable of running simulations with certain parameterCombinations.

```
#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

8.26.1 Detailed Description

A class capable of running simulations with certain parameterCombinations.
Definition at line 38 of file SimulationRunner.h.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 SimulationRunner()

```
SimulationRunner::SimulationRunner (
    unsigned int threads ) [explicit]
```

Creates a [SimulationRunner](#) which can use no more than the given number of threads to simulate parameter↔Combinations concurrently.

Parameters

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

Definition at line 11 of file SimulationRunner.cpp.

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

8.26.3 Member Function Documentation

8.26.3.1 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 19 of file SimulationRunner.cpp.

References `Status::getName()`.

8.26.3.2 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 23 of file SimulationRunner.cpp.

References `Status::getStatus()`.

8.26.3.3 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 27 of file `SimulationRunner.cpp`.

References `Status::getStatusBar()`.

8.26.3.4 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 15 of file `SimulationRunner.cpp`.

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

8.26.3.5 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:

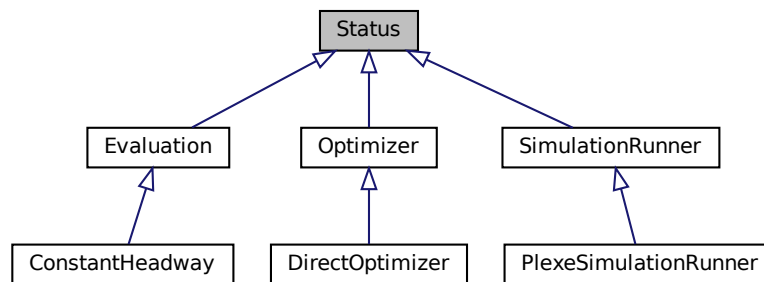
- `src/runner/SimulationRunner.h`
- `src/runner/SimulationRunner.cpp`

8.27 Status Class Reference

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

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

Inheritance diagram for Status:



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.

8.27.1 Detailed Description

An interface defining functions for status updates on configuration and progress of a class. Used for creation of a [StatusBar](#). Overriding the defined methods is not mandatory but recommended. Definition at line 26 of file Status.h.

8.27.2 Member Function Documentation

8.27.2.1 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](#), [Optimizer](#), [DirectOptimizer](#), [Evaluation](#), and [ConstantHeadway](#).

Definition at line 21 of file Status.cpp.

References `NO_NAME`.

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

8.27.2.2 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](#), [Optimizer](#), [DirectOptimizer](#), [Evaluation](#), and [ConstantHeadway](#).

Definition at line 13 of file Status.cpp.

References `NO_STATUS_SUPPORT`.

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

8.27.2.3 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](#), [Optimizer](#), [DirectOptimizer](#), [Evaluation](#), and [ConstantHeadway](#).

Definition at line 17 of file Status.cpp.

References `NO_STATUS_SUPPORT`.

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

8.27.3 Member Data Documentation

8.27.3.1 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 35 of file Status.h.

Referenced by `getName()`.

8.27.3.2 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 31 of file Status.h.

Referenced by [getStatus\(\)](#), and [getStatusBar\(\)](#).

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

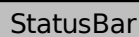
- [src/status/Status.h](#)
- [src/status/Status.cpp](#)

8.28 StatusBar Class Reference

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

```
#include "StatusBar.h"
```

Collaboration diagram for StatusBar:



```
graph TD
    StatusBar[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](#) > > &results, std::pair< [parameterCombination](#), [functionValue](#) > ¤tVal, bool stepChanged=false, [step](#) currentStep=INIT)
Prints the given parameterCombinations and respective values to command line.

Static Private Member Functions

- static void [printResult](#) (const [parameterCombination](#) &cords, [functionValue](#) optimum)
Prints the given result command line.
- static void [printStatus](#) ([Status](#) *object)
Prints the [Status](#) of the given object to the command line using [Status::getStatus](#).

Private Attributes

- 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.

8.28.1 Detailed Description

A class used to conduct command line output containing information about the state of the used [Optimizer](#), [SimulationRunner](#) and [Evaluation](#) along with the found optima.
Definition at line 37 of file `StatusBar.h`.

8.28.2 Member Function Documentation

8.28.2.1 `printResult()`

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

Prints the given result command line.

Parameters

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

Definition at line 55 of file `StatusBar.cpp`.

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

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

8.28.2.2 `printResults()`

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

Prints the given parameterCombinations and respective values to command line.

Parameters

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

Definition at line 74 of file `StatusBar.cpp`.

References [LARGE_DIVIDER](#), [printResult\(\)](#), and [SMALL_DIVIDER](#).

8.28.2.3 `printStatus()`

```
void StatusBar::printStatus (
    Status * object ) [static], [private]
```

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

Parameters

<i>object</i>	Object that inherits from Status and whose state is being printed.
---------------	--

Definition at line 69 of file StatusBar.cpp.

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

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

8.28.2.4 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>	parameterCombination 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 16 of file StatusBar.cpp.

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

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

8.28.3 Member Data Documentation

8.28.3.1 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 42 of file StatusBar.h.

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

8.28.3.2 lastStatus

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

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

Definition at line 59 of file StatusBar.h.

Referenced by `updateStatus()`.

8.28.3.3 lastStep

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

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

Definition at line 55 of file `StatusBar.h`.

Referenced by `updateStatus()`.

8.28.3.4 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 51 of file `StatusBar.h`.

Referenced by `updateStatus()`.

8.28.3.5 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 46 of file `StatusBar.h`.

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

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

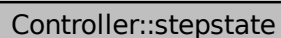
- `src/status/StatusBar.h`
- `src/status/StatusBar.cpp`

8.29 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`:



```
graph TD
    subgraph Controller_stepstate [Controller::stepstate]
    end
```

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.

8.29.1 Detailed Description

A struct keeping track of the currently running optimization step for `StatusBar::updateStatus`.
Definition at line 123 of file `Controller.h`.

8.29.2 Member Function Documentation

8.29.2.1 `get()`

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

Returns the value of `currentStep`.

Returns

The step that is currently run.

Definition at line 145 of file `Controller.h`.

References `currentStep`, and `stepChanged`.

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

8.29.2.2 `next()`

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

Switches `currentStep` to the next step.

Definition at line 136 of file `Controller.h`.

References `currentStep`, and `stepChanged`.

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

8.29.3 Member Data Documentation

8.29.3.1 `currentStep`

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

Current step the optimization is in.

Definition at line 131 of file `Controller.h`.

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

8.29.3.2 `stepChanged`

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

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

Definition at line 127 of file `Controller.h`.

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

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

- `src/controller/Controller.h`

8.30 StoppingCondition Class Reference

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

```
#include "StoppingCondition.h"
```

Collaboration diagram for StoppingCondition:



```
graph TD
    StoppingCondition[StoppingCondition]
```

Public Member Functions

- [StoppingCondition](#) (size_t evaluations=0, size_t hyrects=0, unsigned int minutes=0, [functionValue](#) accuracy=0, unsigned int accuracyIterations=0)
Creates a [StoppingCondition](#) with the given condition values.
- [StoppingCondition](#) (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, size_t hyrects, [functionValue](#) newBestVal)
Checks if any of the configured conditions is met for the given parameters.
- unsigned int [getIterationsSinceImprov](#) () const
Returns the value of [iterationsSinceImprov](#).

Private Member Functions

- bool [updateAccuracy](#) ([functionValue](#) newBestVal)
Checks if the current optimum improves the one saved in [bestVal](#) by more than [ACCURACY](#).

Private Attributes

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

8.30.1 Detailed Description

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

Every conditions is optional and can be set in config. The optimization is stopped when one of the activated conditions is met.

Definition at line 20 of file StoppingCondition.h.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 StoppingCondition() [1/2]

```
StoppingCondition::StoppingCondition (
    size_t evaluations = 0,
    size_t hyrects = 0,
    unsigned int minutes = 0,
    functionValue accuracy = 0,
    unsigned int accuracyIterations = 0 ) [explicit]
```

Creates a [StoppingCondition](#) with the given condition values.

Parameters

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

Definition at line 8 of file StoppingCondition.cpp.

References [ACCURACY](#), mins, [NR_ACCURACY_ITERATIONS](#), [NR_EVALUATIONS](#), [NR_HYRECTS](#), and [time_↔eval](#).

8.30.2.2 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 29 of file StoppingCondition.cpp.

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

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

8.30.3 Member Function Documentation

8.30.3.1 evaluate()

```
bool StoppingCondition::evaluate (
    size_t evaluations,
    size_t hyrects,
    functionValue newBestVal )
```


Checks if any of the configured conditions is met for the given parameters.

Parameters

<i>evaluations</i>	Number of evaluations conducted by the optimization.
<i>hyrects</i>	Number of rectangles in the current partition.
<i>newBestVal</i>	Value of the current optimum.

Returns

A boolean defining whether none of the configured conditions is met (meaning whether the optimization should keep running).

Definition at line 39 of file StoppingCondition.cpp.

References `ACCURACY`, `NR_ACCURACY_ITERATIONS`, `NR_EVALUATIONS`, `NR_HYRECTS`, and `updateAccuracy()`.

8.30.3.2 getIterationsSinceImprov()

```
unsigned int StoppingCondition::getIterationsSinceImprov ( ) const
```

Returns the value of `iterationsSinceImprov`.

Returns

An integral representing the number of iterations since the best value improved by more than `ACCURACY`.

Definition at line 64 of file StoppingCondition.cpp.

References `iterationsSinceImprov`.

8.30.3.3 setStartNow()

```
void StoppingCondition::setStartNow ( )
```

Sets `END_TIME` to be the current time plus `mins`.

Definition at line 47 of file StoppingCondition.cpp.

References `time_eval`.

8.30.3.4 updateAccuracy()

```
bool StoppingCondition::updateAccuracy (
    functionValue newBestVal ) [private]
```

Checks if the current optimum improves the one saved in `bestVal` by more than `ACCURACY`.

If that is the case, `iterationsSinceImprov` is reset to zero and the current optimum is saved in `bestVal`. If not `iterationsSinceImprov` is increased.

Parameters

<i>newBestVal</i>	Current optimum.
-------------------	------------------

Returns

A bool defining if the accuracy condition is met after the values where updated.

Definition at line 54 of file StoppingCondition.cpp.

References `ACCURACY`, `bestVal`, `iterationsSinceImprov`, and `NR_ACCURACY_ITERATIONS`.

Referenced by `evaluate()`.

8.30.4 Member Data Documentation

8.30.4.1 ACCURACY

```
const functionValue StoppingCondition::ACCURACY [private]
```

Accuracy used in accuracy condition.

When the [bestVal](#) has not changed more than [ACCURACY](#) after [NR_ACCURACY_ITERATIONS](#) iterations, the optimization is stopped.

Definition at line 49 of file `StoppingCondition.h`.

Referenced by `StoppingCondition()`, `evaluate()`, and `updateAccuracy()`.

8.30.4.2 bestVal

```
functionValue StoppingCondition::bestVal = INFINITY [private]
```

Best value used to keep track of accuracy condition.

When the [bestVal](#) has not changed more than [ACCURACY](#) after [NR_ACCURACY_ITERATIONS](#) iterations, the optimization is stopped.

Definition at line 59 of file `StoppingCondition.h`.

Referenced by `updateAccuracy()`.

8.30.4.3 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 35 of file `StoppingCondition.h`.

8.30.4.4 iterationsSinceImprov

```
unsigned int StoppingCondition::iterationsSinceImprov = 0 [private]
```

Number of iterations since last improvement of the optimum used to keep track of accuracy condition.

When the [bestVal](#) has not changed more than [ACCURACY](#) after [NR_ACCURACY_ITERATIONS](#) iterations, the optimization is stopped.

Definition at line 64 of file `StoppingCondition.h`.

Referenced by `getIterationsSinceImprov()`, and `updateAccuracy()`.

8.30.4.5 mins

```
const unsigned int StoppingCondition::mins [private]
```

Number of minutes after which the optimization should stop.

Definition at line 39 of file `StoppingCondition.h`.

Referenced by `StoppingCondition()`.

8.30.4.6 NR_ACCURACY_ITERATIONS

```
const unsigned int StoppingCondition::NR_ACCURACY_ITERATIONS [private]
```

Number of iterations used in accuracy condition.

When the [bestVal](#) has not changed more than [ACCURACY](#) after [NR_ACCURACY_ITERATIONS](#) iterations, the optimization is stopped.

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

Referenced by `StoppingCondition()`, `evaluate()`, and `updateAccuracy()`.

8.30.4.7 NR_EVALUATIONS

```
const size_t StoppingCondition::NR_EVALUATIONS [private]
```

Number of evaluations after which the optimization should stop.

Definition at line 25 of file StoppingCondition.h.

Referenced by StoppingCondition(), and evaluate().

8.30.4.8 NR_HYRECTS

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

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

Definition at line 29 of file StoppingCondition.h.

Referenced by StoppingCondition(), and evaluate().

8.30.4.9 time_eval

```
bool StoppingCondition::time_eval [private]
```

Defines whether the time condition should be used.

Definition at line 43 of file StoppingCondition.h.

Referenced by StoppingCondition(), and setStartNow().

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

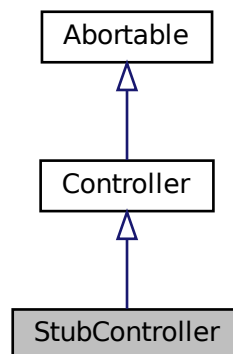
- [src/optimizer/direct/StoppingCondition.h](#)
- [src/optimizer/direct/StoppingCondition.cpp](#)

8.31 StubController Class Reference

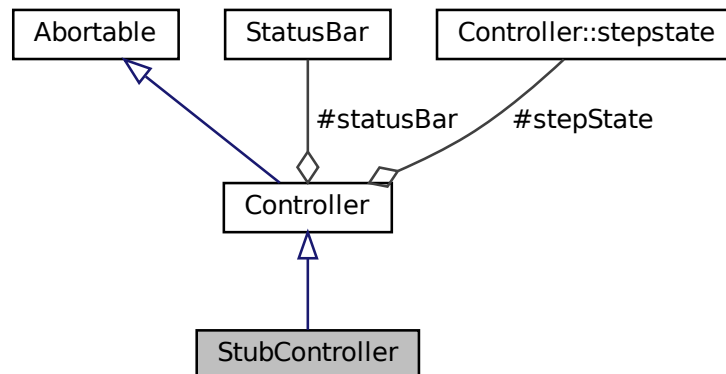
A class that mocks behaviour of [Controller](#).

```
#include "StubController.h"
```

Inheritance diagram for StubController:



Collaboration diagram for StubController:



Public Member Functions

- [StubController](#) (const 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

8.31.1 Detailed Description

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

Instead of real simulations one of the predefined function in [functions](#) is being evaluated, when [Controller::requestValues](#) is called. To use [StubController](#) instead of [Controller](#) a second command line argument has to be passed containing the name of the function to be optimized. The name can be one of the following: quadratic, shekel5, shekel7, shekel10, branin, goldprice, camel6, shubert, hartman3 or hartman6. For more information on all but the first function visit: <https://www.sfu.ca/~ssurjano/optimization.html>
Definition at line 21 of file StubController.h.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 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 142 of file StubController.cpp.
References [Controller::Controller\(\)](#), [f](#), and [functions](#).

8.31.3 Member Function Documentation

8.31.3.1 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 155 of file StubController.cpp.
References [f](#).

8.31.3.2 removeOldResultfiles()

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

Does nothing, since no simulations are run and therefore no result files are created.

Reimplemented from [Controller](#).

Definition at line 164 of file StubController.cpp.

8.31.3.3 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 147 of file StubController.cpp.

8.31.3.4 updateStatus()

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

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

Reimplemented from [Controller](#).

Definition at line 167 of file StubController.cpp.

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

8.31.4 Member Data Documentation

8.31.4.1 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 32 of file StubController.h.

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

8.31.4.2 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, shubert, hartman3 and hartman6.

For more information on all but the first function visit: <https://www.sfu.ca/~ssurjano/optimization.html>

Definition at line 27 of file StubController.h.

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

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

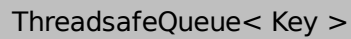
- [src/controller/StubController.h](#)
- [src/controller/StubController.cpp](#)

8.32 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 >:



```
graph TD
    ThreadsafeQueue[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.

8.32.1 Detailed Description

```
template<class Key>
class ThreadsafeQueue< Key >
```

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

Template Parameters

Key	Type of elements in the contained queue.
-----	--

Definition at line 18 of file ThreadsafeQueue.h.

8.32.2 Member Function Documentation

8.32.2.1 getSize()

```
template<class Key >
size_t ThreadsafeQueue< Key >::getSize
Returns current size of the underlying queue structure.
```

Returns

A number representing the size of the queue.

Definition at line 38 of file ThreadsafeQueue.tpp.

References ThreadsafeQueue< Key >::queueLock, and ThreadsafeQueue< Key >::safeQueue.

8.32.2.2 getStartSize()

```
template<class Key >
size_t ThreadsafeQueue< Key >::getStartSize
Returns the value of startSize.
```

Returns

A number representing the number of tasks, when [push](#) was called last.

Definition at line 30 of file ThreadsafeQueue.tpp.

References ThreadsafeQueue< Key >::queueLock, and ThreadsafeQueue< Key >::startSize.

8.32.2.3 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 17 of file ThreadsafeQueue.tpp.

References ThreadsafeQueue< Key >::queueLock, and ThreadsafeQueue< Key >::safeQueue.

8.32.2.4 push()

```
template<class Key >
void ThreadsafeQueue< Key >::push (
    Key val )
```

Adds the given value to [safeQueue](#).

Parameters

<i>val</i>	Values to be added to queue.
------------	------------------------------

Definition at line 9 of file ThreadsafeQueue.tpp.

References ThreadsafeQueue< Key >::queueLock, ThreadsafeQueue< Key >::safeQueue, and ThreadsafeQueue< Key >::startSize.

8.32.3 Member Data Documentation

8.32.3.1 queueLock

```
template<class Key >
```

```
std::mutex ThreadsafeQueue< Key >::queueLock [private]
```

Threadlock to avoid damage to [safeQueue](#) on concurrent access.

Definition at line 27 of file ThreadsafeQueue.h.

Referenced by ThreadsafeQueue< Key >::getSize(), ThreadsafeQueue< Key >::getStartSize(), ThreadsafeQueue< Key >::pop(), and ThreadsafeQueue< Key >::push().

8.32.3.2 safeQueue

```
template<class Key >
```

```
std::queue<Key> ThreadsafeQueue< Key >::safeQueue [private]
```

The actual queue data structure.

Definition at line 23 of file ThreadsafeQueue.h.

Referenced by ThreadsafeQueue< Key >::getSize(), ThreadsafeQueue< Key >::pop(), and ThreadsafeQueue< Key >::push().

8.32.3.3 startSize

```
template<class Key >
```

```
size_t ThreadsafeQueue< Key >::startSize = 0 [private]
```

Number of elements in queue when [push](#) was called the last time.

Can be used for progress information.

Definition at line 32 of file ThreadsafeQueue.h.

Referenced by ThreadsafeQueue< Key >::getStartSize(), and ThreadsafeQueue< Key >::push().

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

- src/Utils/ThreadsafeQueue.h
- src/Utils/ThreadsafeQueue.hpp

8.33 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`.

8.33.1 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. Definition at line 28 of file ValueMap.h.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 ValueMap()

```
ValueMap::ValueMap (
    unsigned int topEntries = 10 ) [explicit]
```

Creates a [ValueMap](#).

Parameters

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

Definition at line 12 of file ValueMap.cpp.

References [topEntries](#).

8.33.3 Member Function Documentation

8.33.3.1 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 50 of file ValueMap.cpp.

References [topEntries](#), [topVals](#), and [values](#).

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

8.33.3.2 getMedian()

```
functionValue ValueMap::getMedian ( )
```

Returns the median of all values using [lowerValues](#) and [upperValues](#).

If no values have been added, 0 is returned. Triggers [updateMap](#).

Returns

A value representing the median of all values.

Definition at line 97 of file ValueMap.cpp.

References [getSize\(\)](#), [lowerValues](#), [operationsLock](#), [updateMap\(\)](#), and [upperValues](#).

8.33.3.3 getSize()

```
size_t ValueMap::getSize ( ) const
```

Returns the number of inserted values.

Values in [tba](#) are included.

Returns

An integral representing the number of inserted values.

Definition at line 109 of file ValueMap.cpp.

References [tba](#), and [values](#).

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

8.33.3.4 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 113 of file ValueMap.cpp.

References [topVals](#), and [updateMap\(\)](#).

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

8.33.3.5 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 138 of file ValueMap.cpp.

References [updateMap\(\)](#), and [values](#).

8.33.3.6 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 82 of file ValueMap.cpp.

References [tba](#).

8.33.3.7 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 86 of file ValueMap.cpp.

References `getSize()`, `operationsLock`, `updateMap()`, and `values`.

8.33.3.8 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 118 of file ValueMap.cpp.

References `Parameter::getVal()`, `topVals`, and `updateMap()`.

8.33.3.9 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 71 of file ValueMap.cpp.

References `operationsLock`, `updateMap()`, and `values`.

8.33.3.10 updateMap()

```
void ValueMap::updateMap ( ) [private]
```

Takes all values in `tba`, adds them to `lowerValues` or `upperValues` and inserts them into `values`.

`lowerValues` and `upperValues` are sorted as is required by their constraints. Afterwards `tba` is cleared.

Definition at line 15 of file ValueMap.cpp.

References `addValue()`, `lowerValues`, `operationsLock`, `tba`, and `upperValues`.

Referenced by `getMedian()`, `getTopVals()`, `getValues()`, `isKnown()`, `isTopValue()`, and `query()`.

8.33.4 Member Data Documentation

8.33.4.1 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 42 of file ValueMap.h.

Referenced by [getMedian\(\)](#), and [updateMap\(\)](#).

8.33.4.2 operationsLock

```
std::mutex ValueMap::operationsLock [private]
```

Threadlock to avoid damage to the data structure when concurrent threads access it.

Definition at line 33 of file ValueMap.h.

Referenced by [getMedian\(\)](#), [isKnown\(\)](#), [query\(\)](#), and [updateMap\(\)](#).

8.33.4.3 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 64 of file ValueMap.h.

Referenced by [getSize\(\)](#), [insert\(\)](#), and [updateMap\(\)](#).

8.33.4.4 topEntries

```
const unsigned int ValueMap::topEntries [private]
```

Number of entries to be printed as best values at the end of the optimization process.

Can be configured in main config.

Definition at line 48 of file ValueMap.h.

Referenced by [ValueMap\(\)](#), and [addValue\(\)](#).

8.33.4.5 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 53 of file ValueMap.h.

Referenced by [addValue\(\)](#), [getTopVals\(\)](#), and [isTopValue\(\)](#).

8.33.4.6 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 38 of file ValueMap.h.

Referenced by [getMedian\(\)](#), and [updateMap\(\)](#).

8.33.4.7 values

```
std::map<parameterCombination, functionValue, CmpVectorSharedParameter> ValueMap::values  
[private]
```

Actual map that contains parameterCombinations and their respective values.

Definition at line 58 of file ValueMap.h.

Referenced by `addValue()`, `getSize()`, `getValues()`, `isKnown()`, and `query()`.
The documentation for this class was generated from the following files:

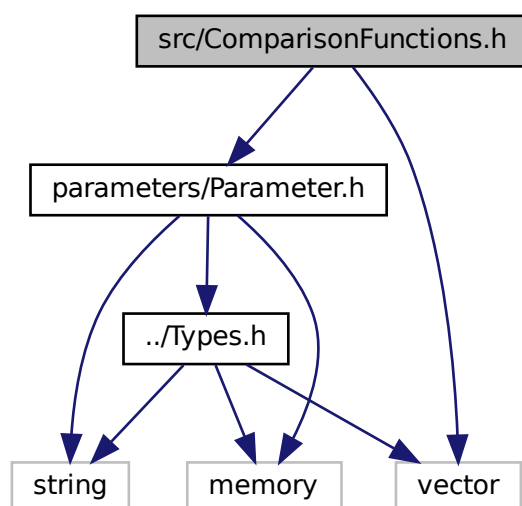
- `src/controller/ValueMap.h`
- `src/controller/ValueMap.cpp`

File Documentation

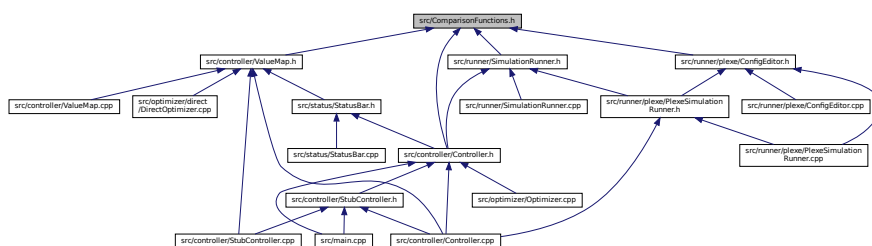
In this file, comparison functions are defined which should be used across the whole framework.

```
#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

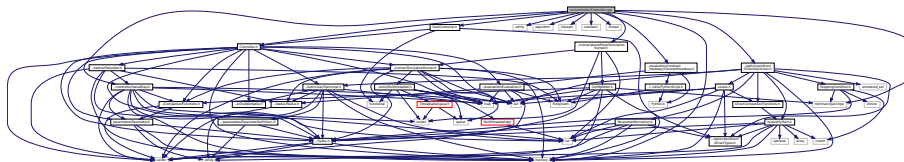
In this file, comparison functions are defined which should be used across the whole framework. They can be used to order elements in STL containers.

9.2 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 "../runner/plexo/PlexoSimulationRunner.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.

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.

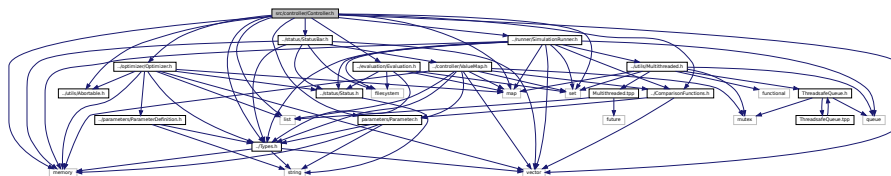
Definition at line 28 of file Controller.cpp.

9.3 src/controller/Controller.h File Reference

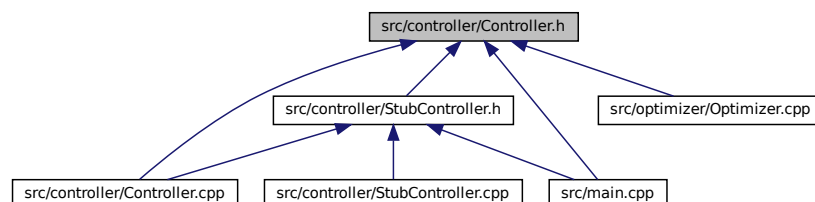
In this file, the header of the [Controller](#) class is defined.

```
#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](#)

Parameters

P	Matrix P used in calculation of Hartman function.
v	

Returns

Definition at line 47 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

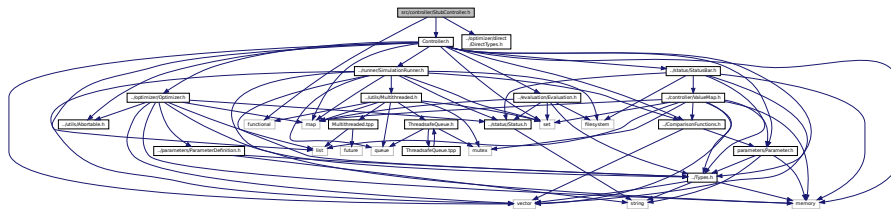
The value of the shekel function at the given argument vector.

Definition at line 20 of file StubController.cpp.

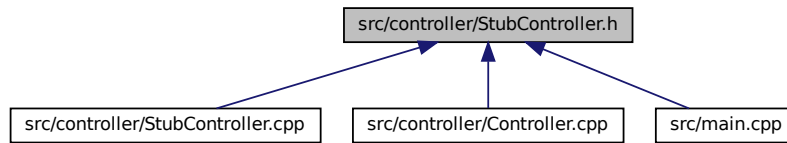
9.5 src/controller/StubController.h File Reference

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

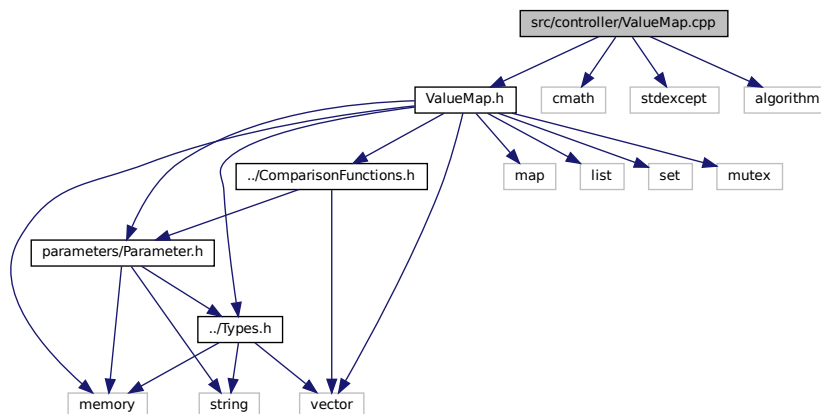
In this file, the header of the [StubController](#) class is defined.

9.6 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

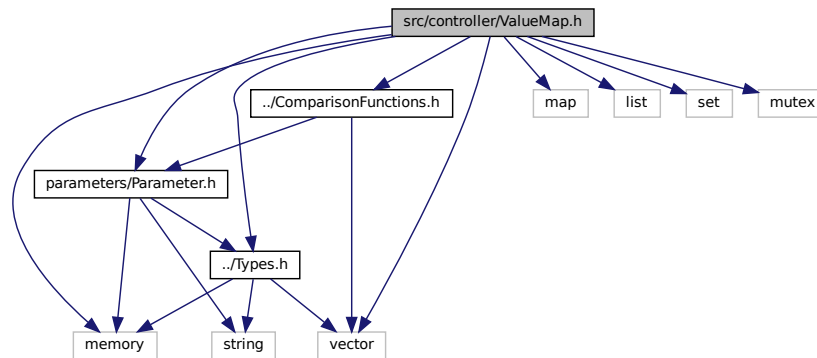
In this file, the implementation of the [ValueMap](#) class is defined.

9.7 src/controller/ValueMap.h File Reference

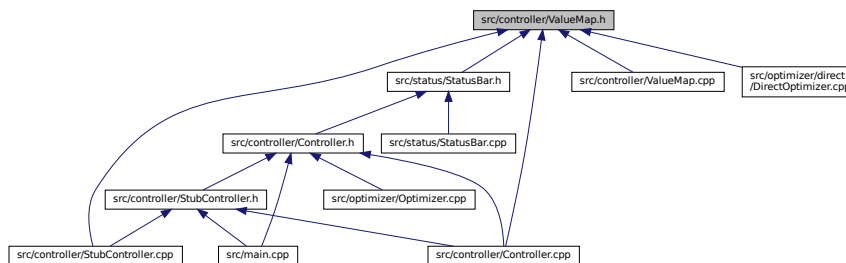
In this file, the header of the [ValueMap](#) class is defined.

```
#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 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.

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 31 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 69 of file constant_headway.py.

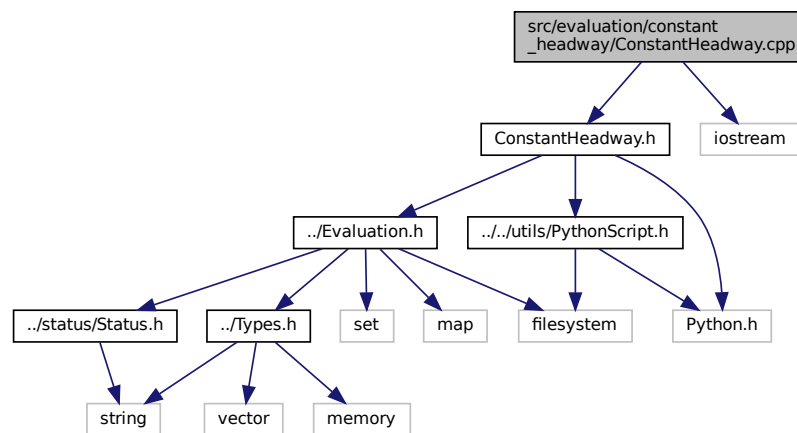
9.9 src/evaluation/constant_headway/ConstantHeadway.cpp File Reference

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.

9.10 src/evaluation/constant_headway/ConstantHeadway.h File Reference

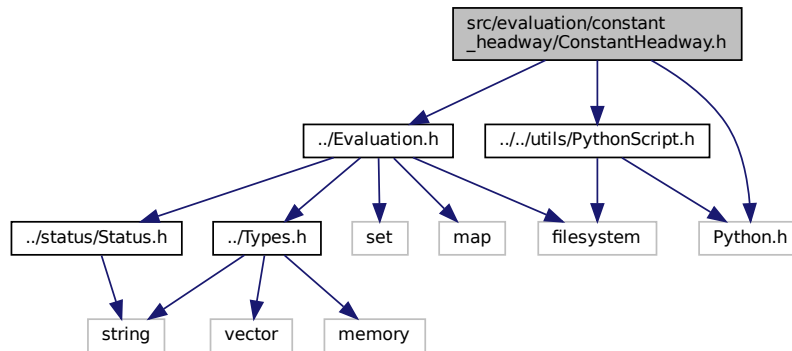
In this file, the header of the [ConstantHeadway](#) class is defined.

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

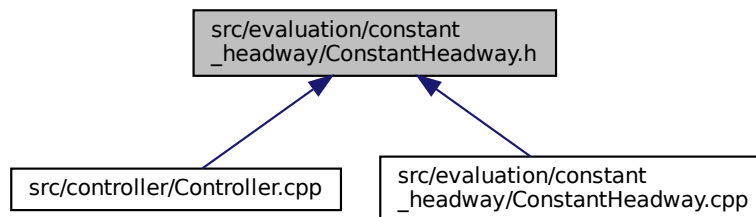
```
#include "../../utils/PythonScript.h"
```

```
#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

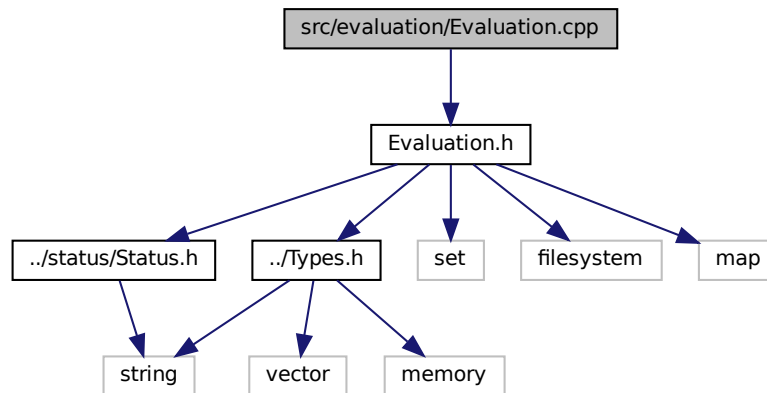
In this file, the header of the [ConstantHeadway](#) class is defined.

9.11 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

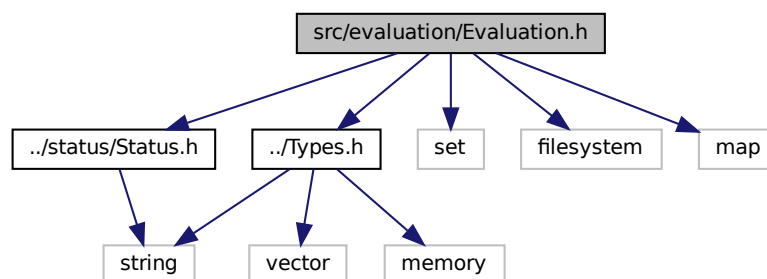
In this file, the implementation of the [Evaluation](#) class is defined.

9.12 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:



9.13.1 Detailed Description

In this file, the main function running the *Simopticon* framework is defined.

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 28 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 43 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.

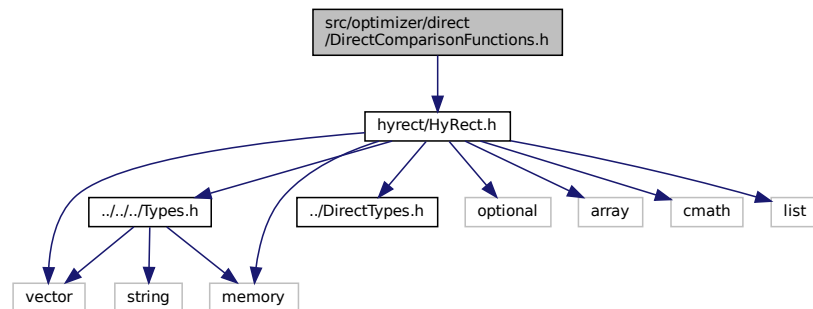
Definition at line 22 of file main.cpp.

9.14 src/optimizer/direct/DirectComparisonFunctions.h File Reference

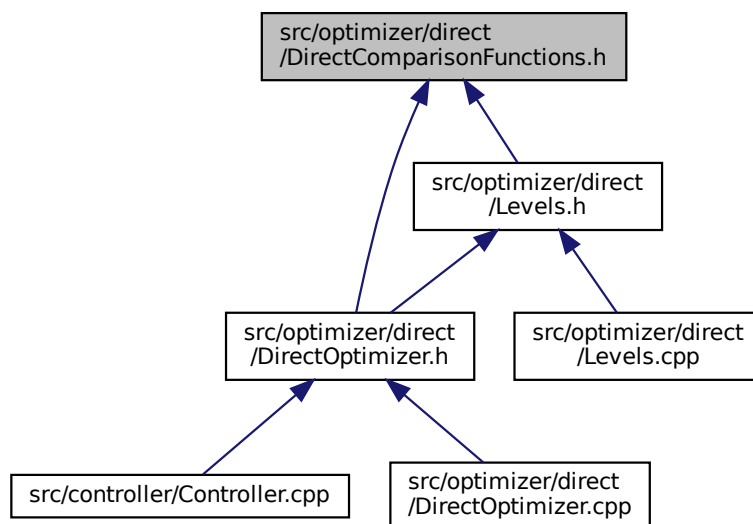
In this file, comparison functions are defined which are used in the direct module.

```
#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

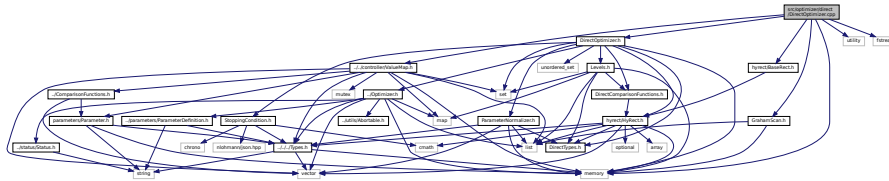
In this file, comparison functions are defined which are used in the direct module. They can be used to order elements in STL containers.

9.15 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

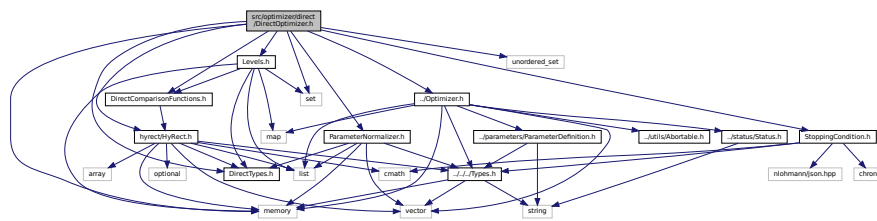
In this file, the implementation of the [DirectOptimizer](#) class is defined.

9.16 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 "StoppingCondition.h"
#include "hyrect/HyRect.h"
#include "ParameterNormalizer.h"
#include "Levels.h"
#include <set>
#include <unordered_set>
#include <memory>
```

Include dependency graph for DirectOptimizer.h:



9.17.2 Typedef Documentation

9.17.2.1 depth

```
typedef unsigned int depth
```

An integral type used for representing the depth of a [HyRect](#) in the partition tree.

Definition at line 14 of file DirectTypes.h.

9.17.2.2 dimension

```
typedef unsigned char dimension
```

An integral type used for representing a dimension of the search space.

Please note that the first dimension is represented by value 1, not 0.

Definition at line 20 of file DirectTypes.h.

9.17.2.3 dirCoordinate

```
typedef long double dirCoordinate
```

A floating point type used for representing one coordinate in the hypercube search space.

Definition at line 25 of file DirectTypes.h.

9.18 src/optimizer/direct/GrahamScan.cpp File Reference

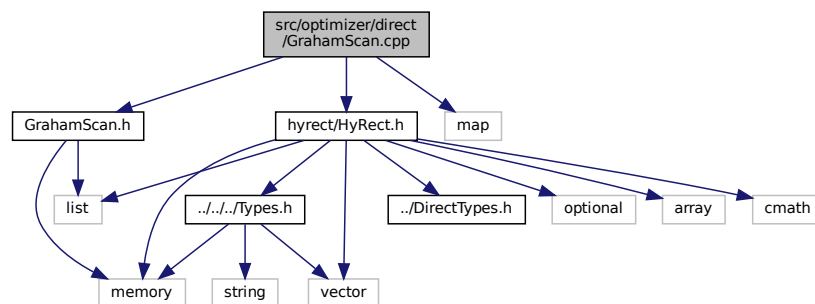
In this file, the implementation of the [GrahamScan](#) class is defined.

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

```
#include "hyrect/HyRect.h"
```

```
#include <map>
```

Include dependency graph for GrahamScan.cpp:



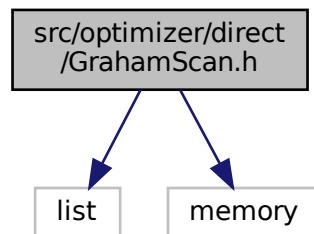
9.18.1 Detailed Description

In this file, the implementation of the [GrahamScan](#) class is defined.

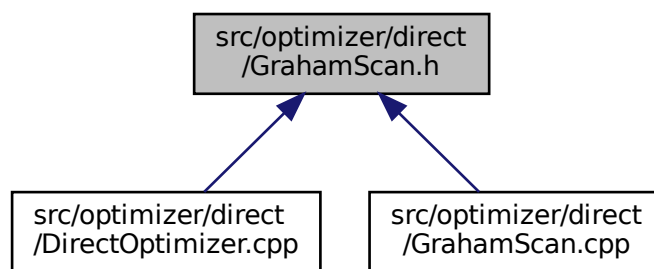
9.19 src/optimizer/direct/GrahamScan.h File Reference

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.

9.19.1 Detailed Description

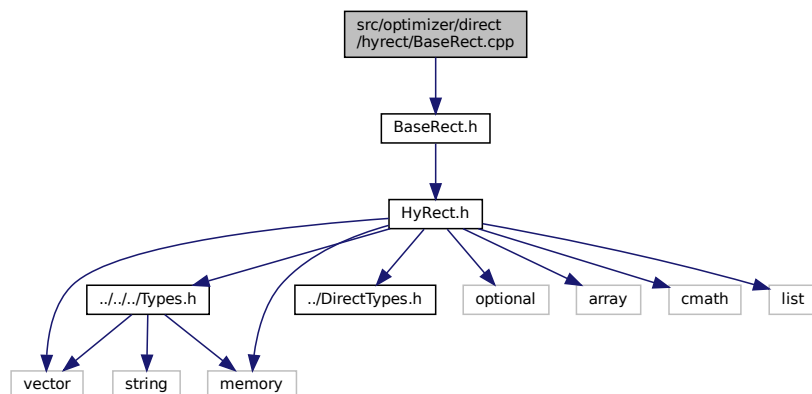
In this file, the header of the [GrahamScan](#) class is defined.

9.20 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:



9.20.1 Detailed Description

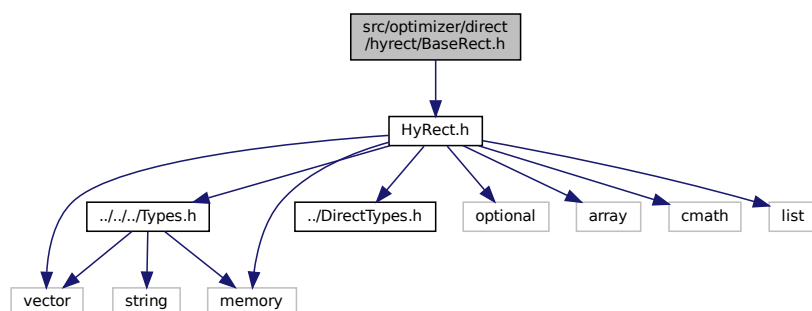
In this file, the implementation of the [BaseRect](#) class is defined.

9.21 src/optimizer/direct/hyrect/BaseRect.h File Reference

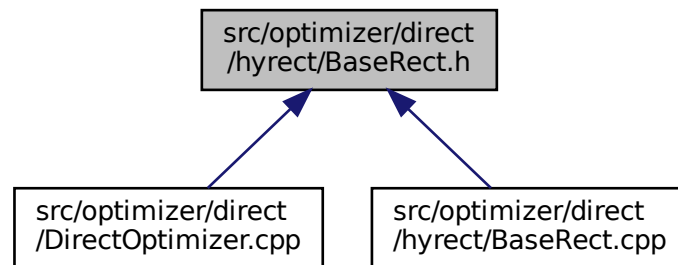
In this file, the header of the [BaseRect](#) class is defined.

```
#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.

9.21.1 Detailed Description

In this file, the header of the [BaseRect](#) class is defined.

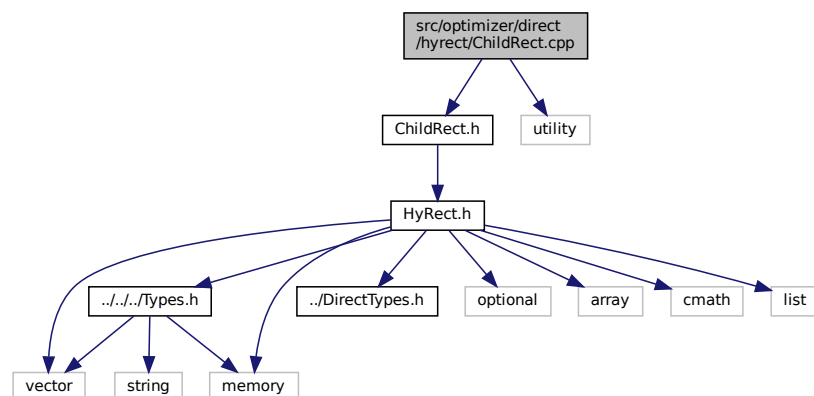
9.22 src/optimizer/direct/hyrect/ChildRect.cpp File Reference

In this file, the implementation of the [ChildRect](#) class is defined.

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

```
#include <utility>
```

Include dependency graph for `ChildRect.cpp`:



9.22.1 Detailed Description

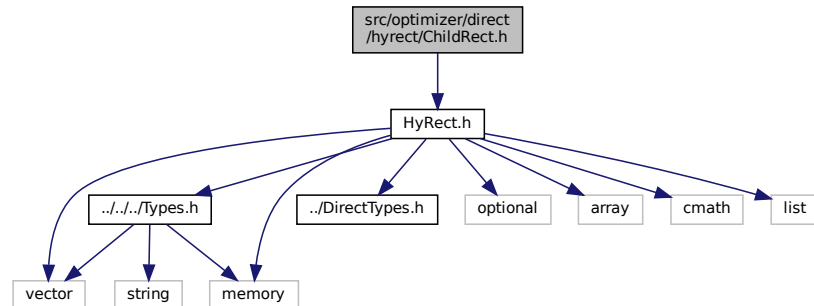
In this file, the implementation of the [ChildRect](#) class is defined.

9.23 src/optimizer/direct/hyrect/ChildRect.h File Reference

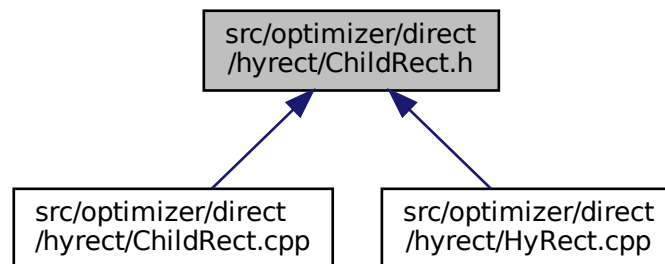
In this file, the header of the [ChildRect](#) class is defined.

```
#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](#).

9.23.1 Detailed Description

In this file, the header of the [ChildRect](#) class is defined.

9.24 src/optimizer/direct/hyrect/HyRect.cpp File Reference

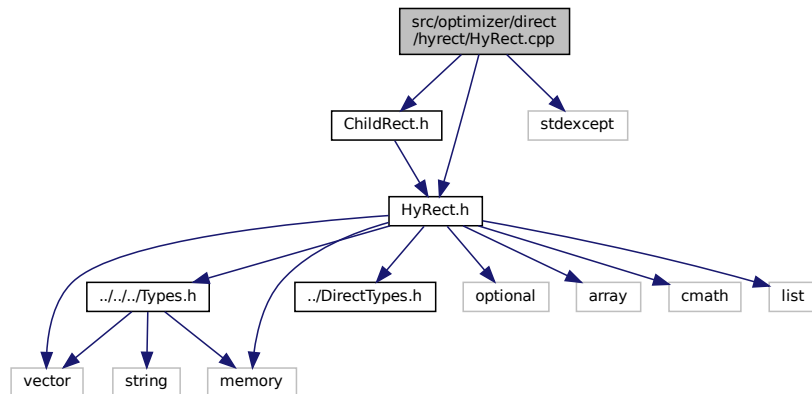
In this file, the implementation of the [HyRect](#) class is defined.

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

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

```
#include <stdexcept>
```

Include dependency graph for HyRect.cpp:



9.24.1 Detailed Description

In this file, the implementation of the [HyRect](#) class is defined.

9.25 src/optimizer/direct/hyrect/HyRect.h File Reference

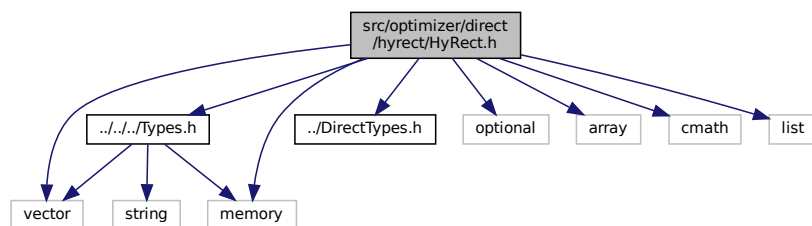
In this file, the header of the [HyRect](#) class is defined.

```

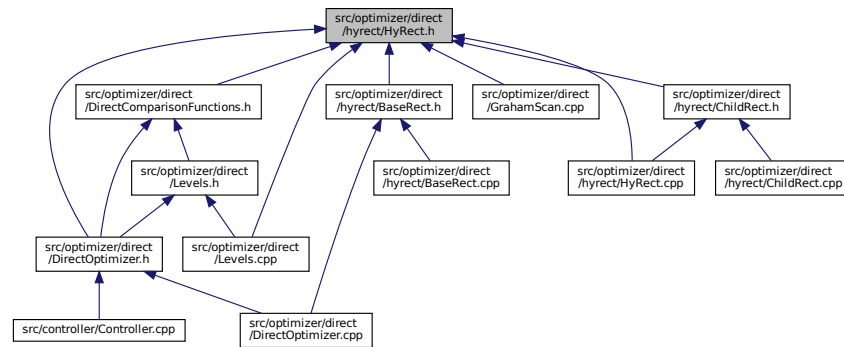
#include "../../Types.h"
#include "../DirectTypes.h"
#include <optional>
#include <array>
#include <vector>
#include <cmath>
#include <list>
#include <memory>

```

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](#).

9.25.1 Detailed Description

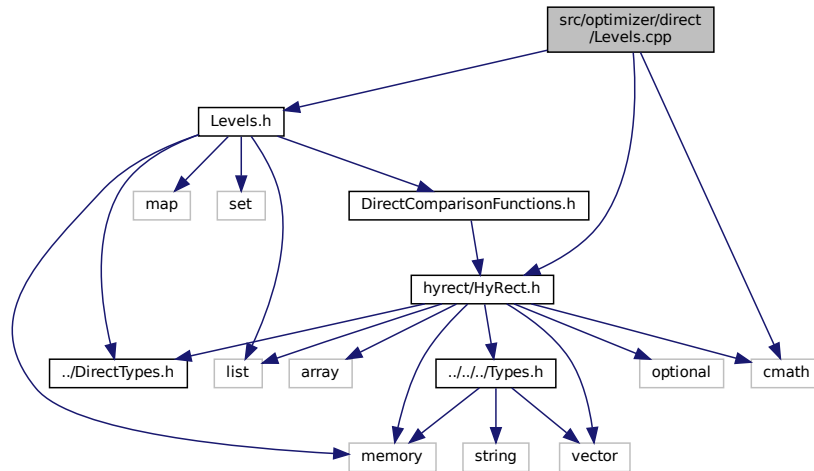
In this file, the header of the [HyRect](#) class is defined.

9.26 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:



9.26.1 Detailed Description

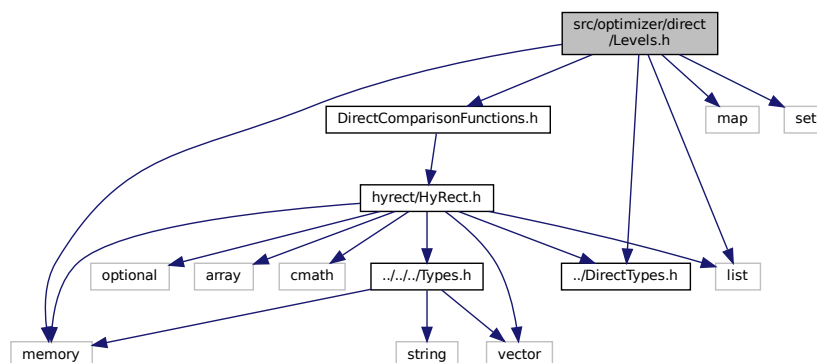
In this file, the implementation of the [Levels](#) class is defined.

9.27 src/optimizer/direct/Levels.h File Reference

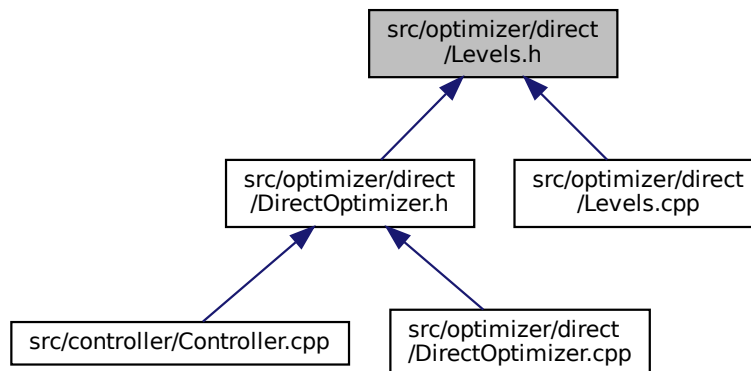
In this file, the header of the [Levels](#) class is defined.

```
#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.

9.27.1 Detailed Description

In this file, the header of the [Levels](#) class is defined.

9.28 src/optimizer/direct/ParameterNormalizer.cpp File Reference

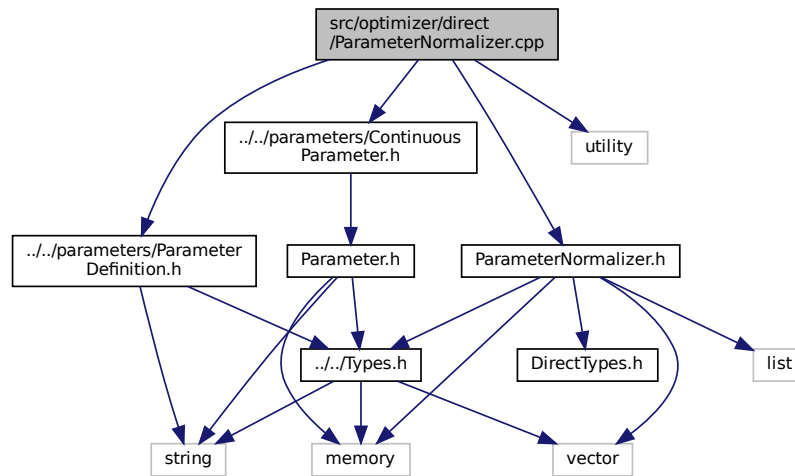
In this file, the implementation of the [ParameterNormalizer](#) class is defined.

```

#include "ParameterNormalizer.h"
#include <utility>
#include "../../parameters/ContinuousParameter.h"
#include "../../parameters/ParameterDefinition.h"

```

Include dependency graph for ParameterNormalizer.cpp:



9.28.1 Detailed Description

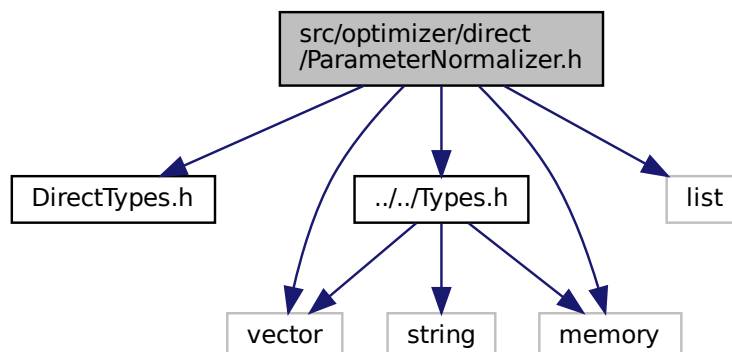
In this file, the implementation of the [ParameterNormalizer](#) class is defined.

9.29 src/optimizer/direct/ParameterNormalizer.h File Reference

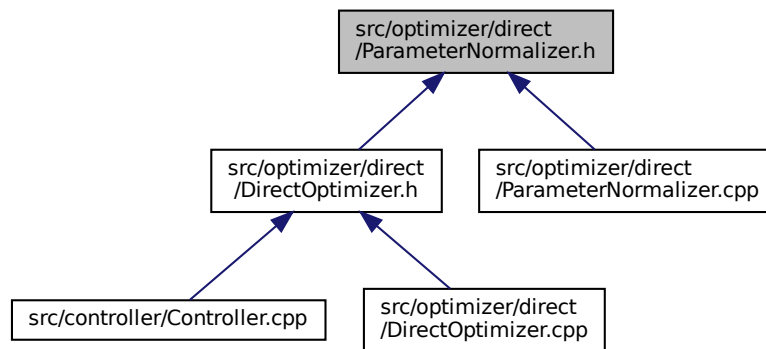
In this file, the header of the [ParameterNormalizer](#) class is defined.

```
#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.

9.29.1 Detailed Description

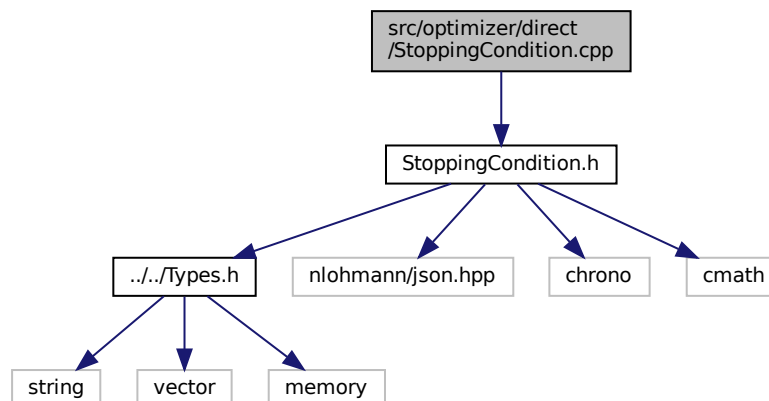
In this file, the header of the [ParameterNormalizer](#) class is defined.

9.30 src/optimizer/direct/StoppingCondition.cpp File Reference

In this file, the implementation of the [StoppingCondition](#) class is defined.

```
#include "StoppingCondition.h"
```

Include dependency graph for `StoppingCondition.cpp`:



Functions

- `template<typename T >`
`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 thats the case.

9.30.1 Detailed Description

In this file, the implementation of the [StoppingCondition](#) class is defined.

9.30.2 Function Documentation

9.30.2.1 getConditionFromJSON()

```
template<typename T >
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 thats 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.

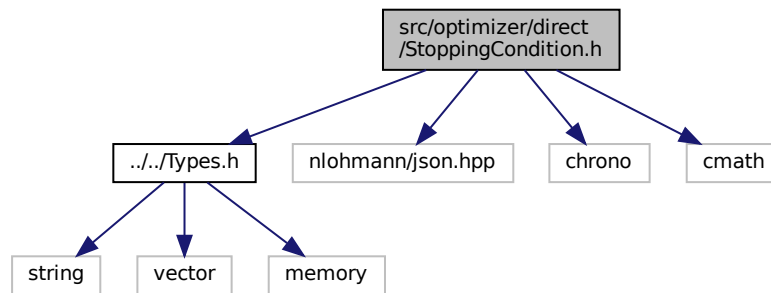
Definition at line 25 of file `StoppingCondition.cpp`.

9.31 src/optimizer/direct/StoppingCondition.h File Reference

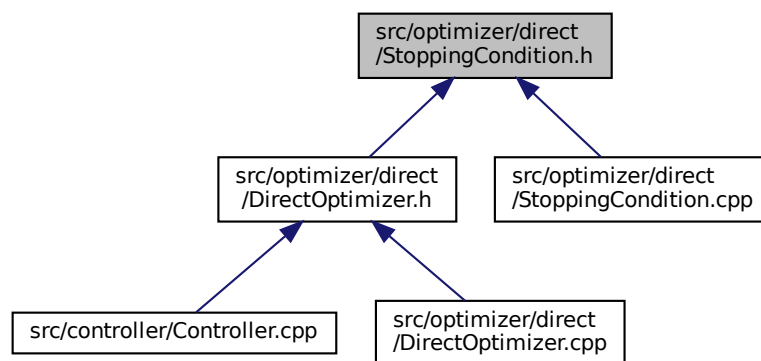
In this file, the header of the [StoppingCondition](#) class is defined.

```
#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 DIRECT should be stopped.

9.31.1 Detailed Description

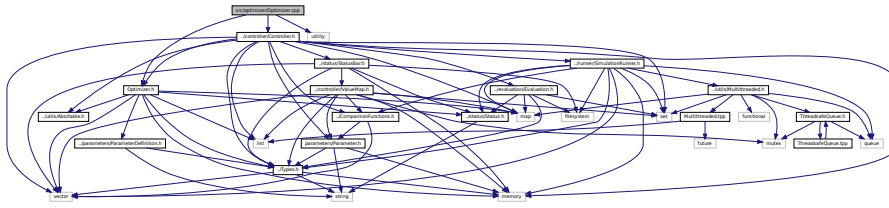
In this file, the header of the [StoppingCondition](#) class is defined.

9.32 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`:



9.32.1 Detailed Description

In this file, the implementation of the `Optimizer` class is defined.

9.33 `src/optimizer/Optimizer.h` File Reference

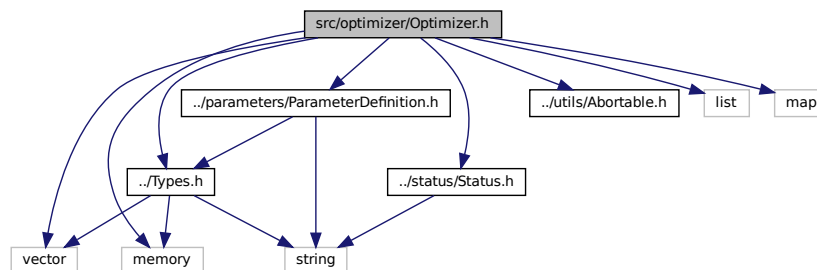
In this file, the header of the `Optimizer` class is defined.

```

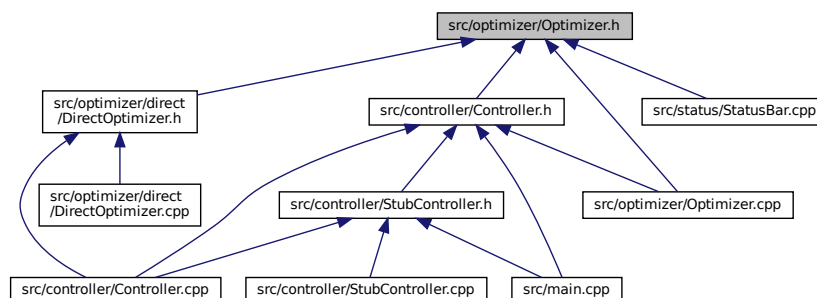
#include "../Types.h"
#include "../parameters/ParameterDefinition.h"
#include "../status/Status.h"
#include "../utils/Abortable.h"
#include <list>
#include <vector>
#include <map>
#include <memory>

```

Include dependency graph for `Optimizer.h`:



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



Classes

- class [Optimizer](#)

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

9.33.1 Detailed Description

In this file, the header of the [Optimizer](#) class is defined.

9.34 src/parameters/ContinuousParameter.cpp File Reference

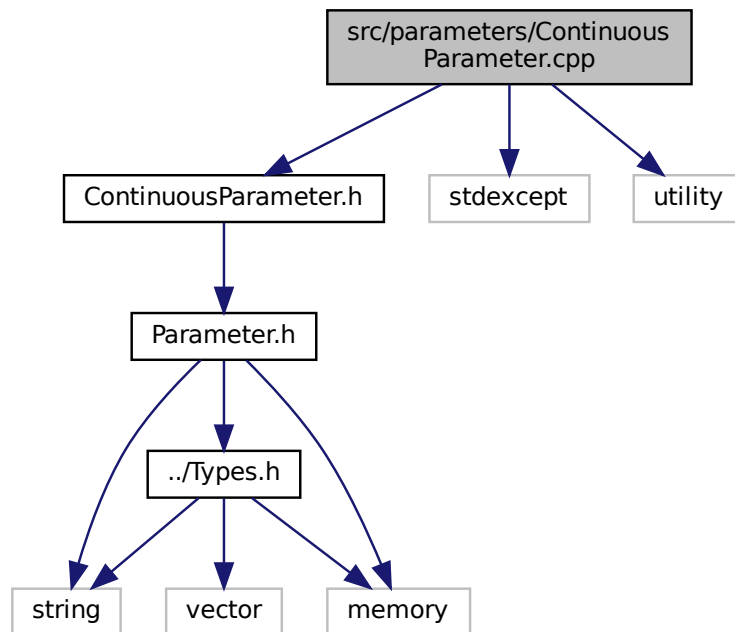
In this file, the implementation of the [ContinuousParameter](#) class is defined.

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

```
#include <stdexcept>
```

```
#include <utility>
```

Include dependency graph for ContinuousParameter.cpp:



9.34.1 Detailed Description

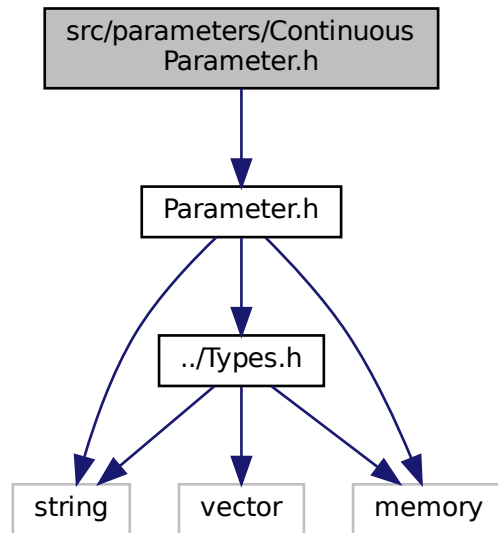
In this file, the implementation of the [ContinuousParameter](#) class is defined.

9.35 src/parameters/ContinuousParameter.h File Reference

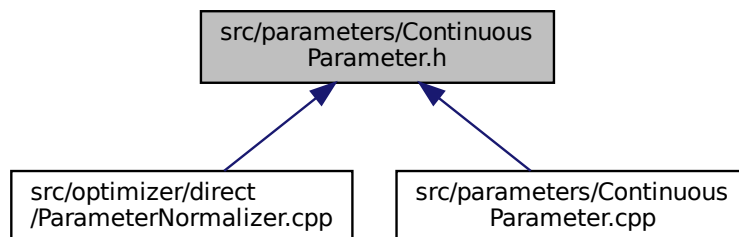
In this file, the header of the [ContinuousParameter](#) class is defined.

```
#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.

9.35.1 Detailed Description

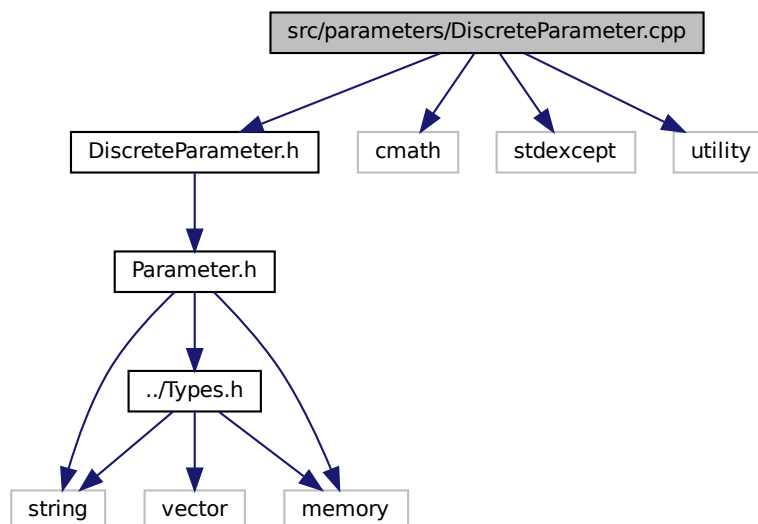
In this file, the header of the [ContinuousParameter](#) class is defined.

9.36 `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:



9.36.1 Detailed Description

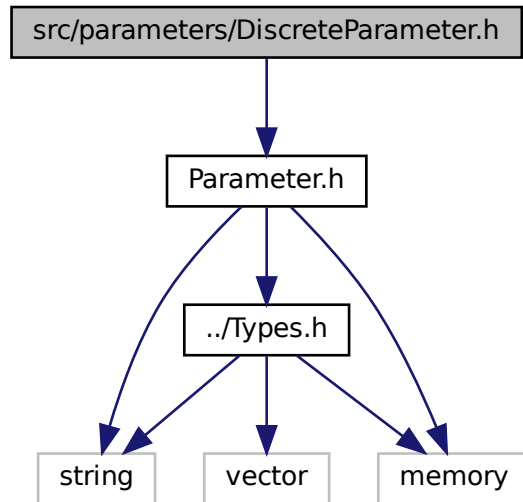
In this file, the implementation of the [DiscreteParameter](#) class is defined.

9.37 src/parameters/DiscreteParameter.h File Reference

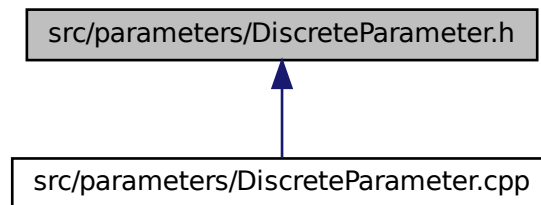
In this file, the header of the [DiscreteParameter](#) class is defined.

```
#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.

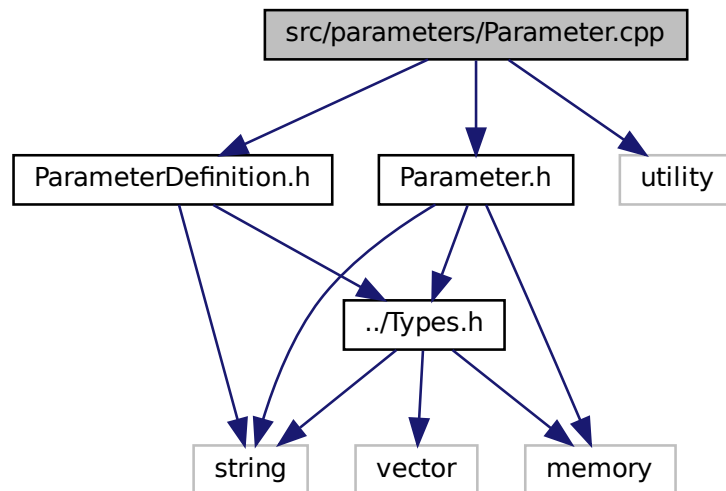
9.37.1 Detailed Description

In this file, the header of the [DiscreteParameter](#) class is defined.

9.38 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:
```



9.38.1 Detailed Description

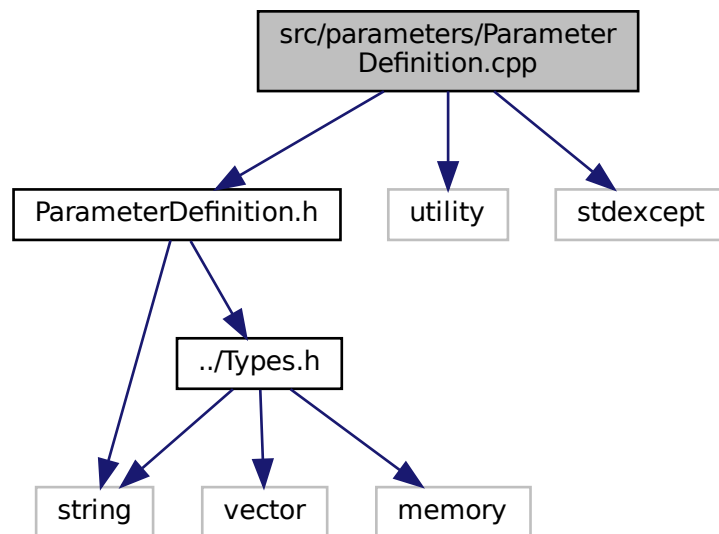
In this file, the implementation of the [Parameter](#) class is defined.

9.39 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 ParameterDefinition.cpp:



9.40.1 Detailed Description

In this file, the implementation of the [ParameterDefinition](#) class is defined.

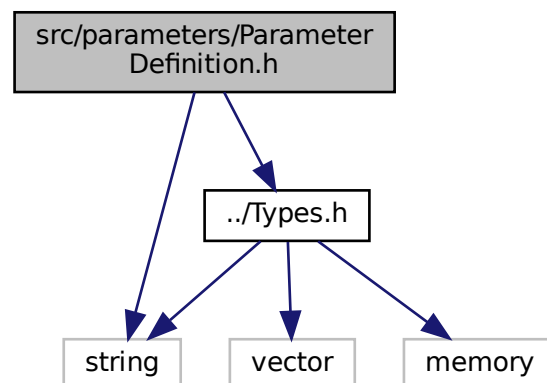
9.41 src/parameters/ParameterDefinition.h File Reference

In this file, the header of the [ParameterDefinition](#) class is defined.

```
#include "../Types.h"
```

```
#include <string>
```

Include dependency graph for `ParameterDefinition.h`:

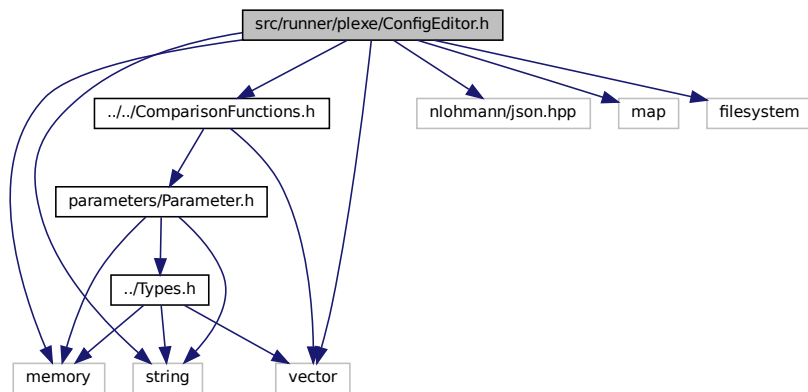


9.43 src/runner/plex/ConfigEditor.h File Reference

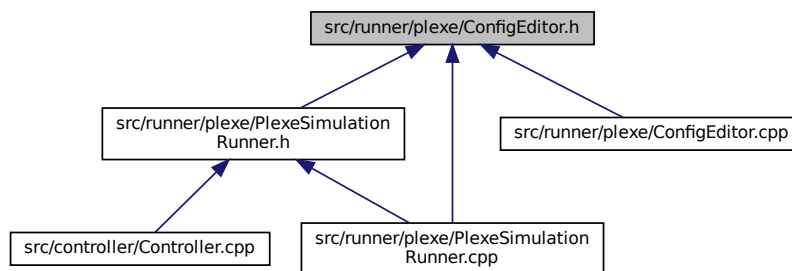
In this file, the header of the [ConfigEditor](#) class is defined.

```
#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

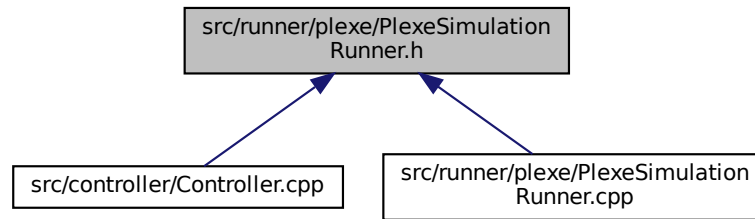
- class [ConfigEditor](#)

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

9.43.1 Detailed Description

In this file, the header of the [ConfigEditor](#) class is defined.

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



Classes

- class [PlexeSimulationRunner](#)

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

9.45.1 Detailed Description

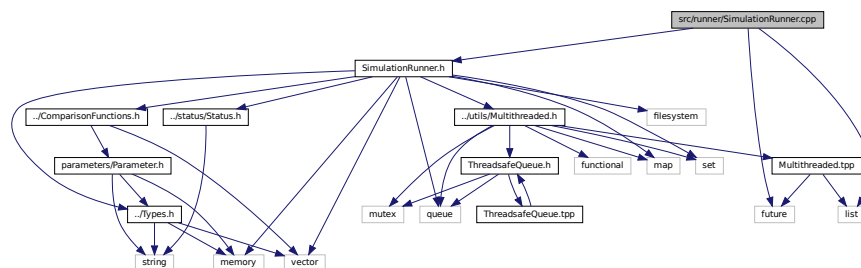
In this file, the header of the [PlexeSimulationRunner](#) class is defined.

9.46 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:



9.46.1 Detailed Description

In this file, the implementation of the [SimulationRunner](#) class is defined.

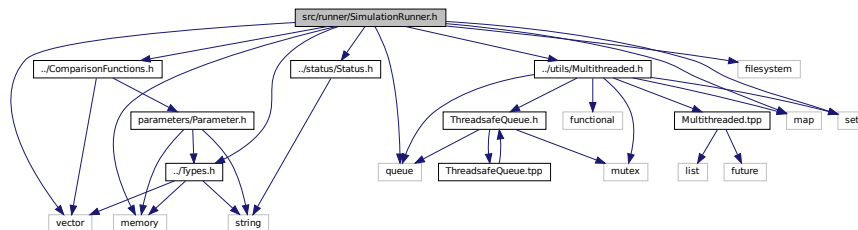
9.47 src/runner/SimulationRunner.h File Reference

In this file, the header of the [SimulationRunner](#) class is defined.

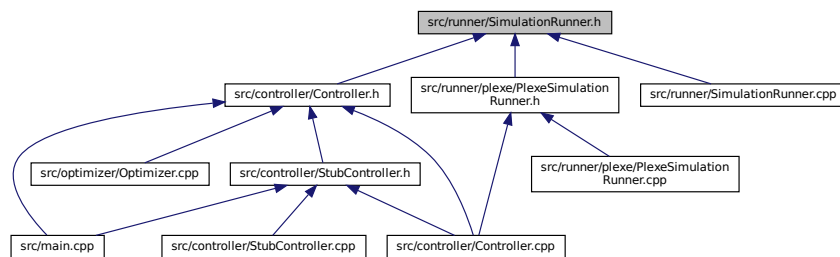
```
#include "../Types.h"
#include "../ComparisonFunctions.h"
#include "../utils/Multithreaded.h"
#include "../status/Status.h"
```

```
#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.

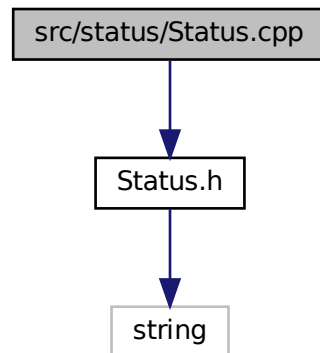
9.47.1 Detailed Description

In this file, the header of the [SimulationRunner](#) class is defined.

9.48 src/status/Status.cpp File Reference

In this file, the implementation of the [Status](#) class is defined.

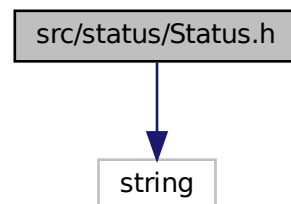
Include dependency graph for Status.cpp:



In this file, the implementation of the `Status` class is defined.

In this file, the header of the `Status` class is defined.

Include dependency graph for Status.h:



Classes

- class [Status](#)

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

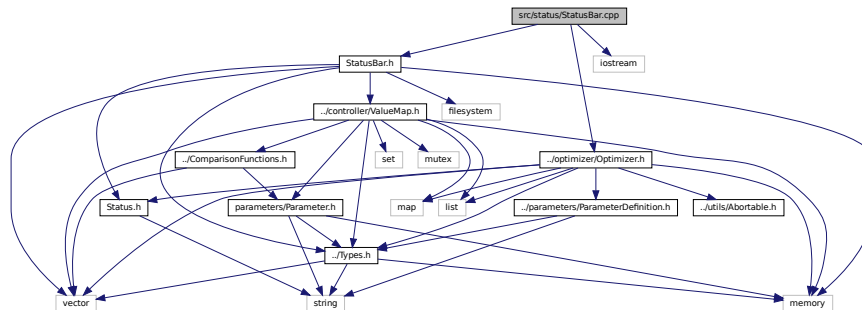
9.49.1 Detailed Description

In this file, the header of the [Status](#) class is defined.

9.50 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:
```



9.50.1 Detailed Description

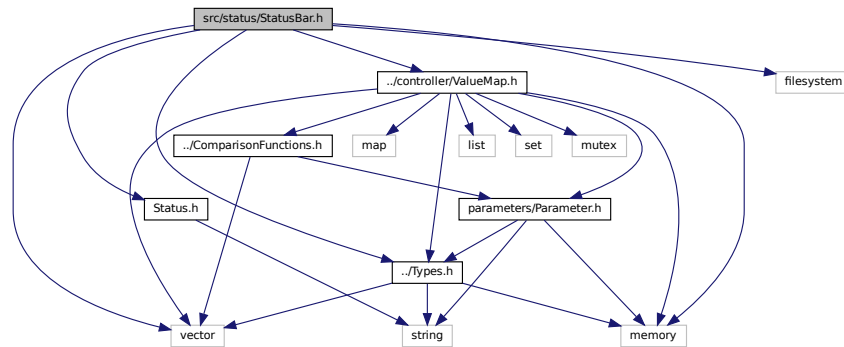
In this file, the implementation of the [StatusBar](#) class is defined.

9.51 src/status/StatusBar.h File Reference

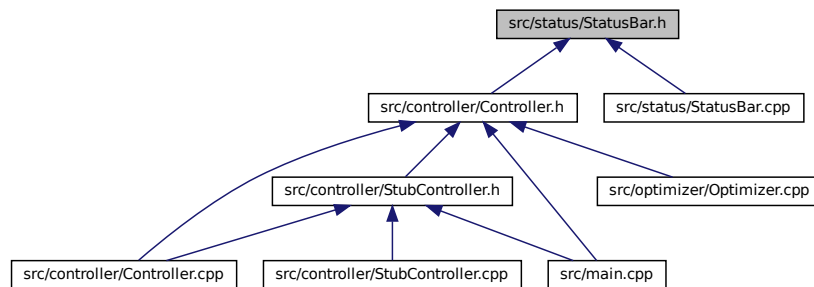
In this file, the header of the [StatusBar](#) class is defined.

```
#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.

9.51.1 Detailed Description

In this file, the header of the [StatusBar](#) class is defined.

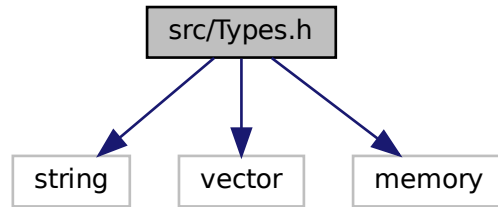
9.52 src/Types.h File Reference

In this file, types are defined which should be used across the whole framework.

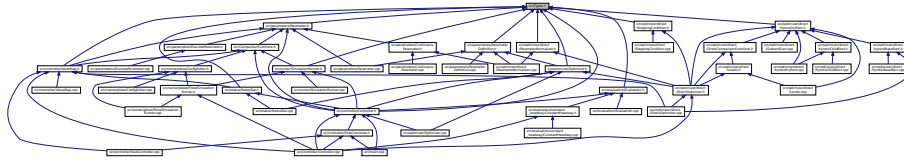
```
#include <string>
#include <vector>
```

```
#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.

9.52.1 Detailed Description

In this file, types are defined which should be used across the whole framework.

9.52.2 Typedef Documentation

9.52.2.1 `coordinate`

```
typedef double coordinate
```

A floating point type used to represent [Parameter](#) values.

Definition at line 31 of file Types.h.

9.52.2.2 functionValue

```
typedef long double functionValue
```

A floating point type containing the value of an optimized function.

Definition at line 26 of file Types.h.

9.52.2.3 parameterCombination

```
typedef std::vector<std::shared_ptr<Parameter> > parameterCombination
```

A complex type representing a vector in parameter space.

Definition at line 21 of file Types.h.

9.52.2.4 runId

```
typedef std::string runId
```

An identifier that makes different simulation runs in one result file folder distinguishable.

Uniqueness is not being asserted.

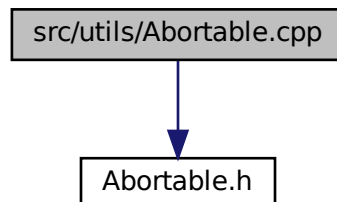
Definition at line 37 of file Types.h.

9.53 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:



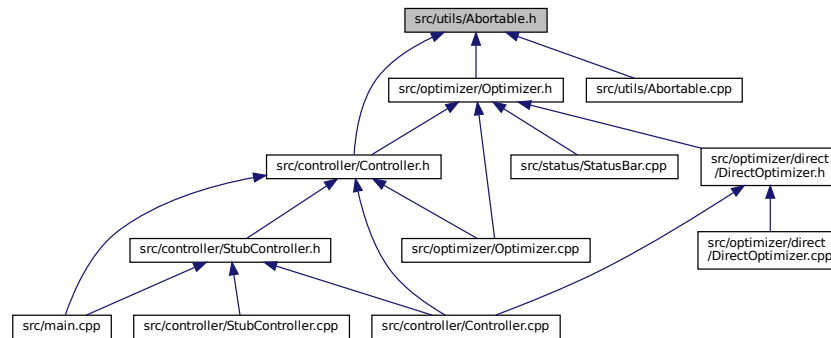
9.53.1 Detailed Description

In this file, the implementation of the [Abortable](#) class is defined.

9.54 src/utils/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.

9.54.1 Detailed Description

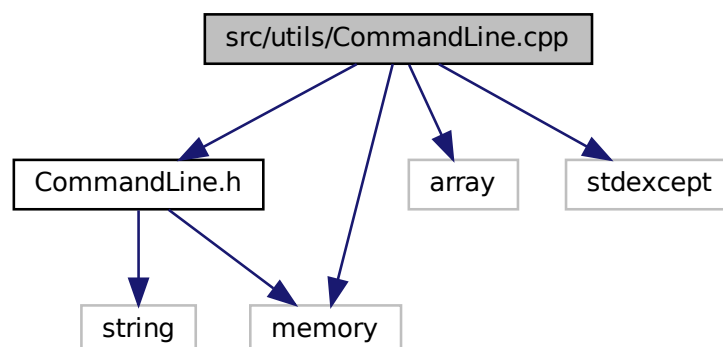
In this file, the header of the [Abortable](#) class is defined.

9.55 src/opts/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:



9.55.1 Detailed Description

In this file, the implementation of the [CommandLine](#) class is defined.

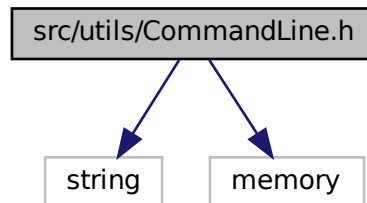
9.56 src/utils/CommandLine.h File Reference

In this file, the header of the [CommandLine](#) class is defined.

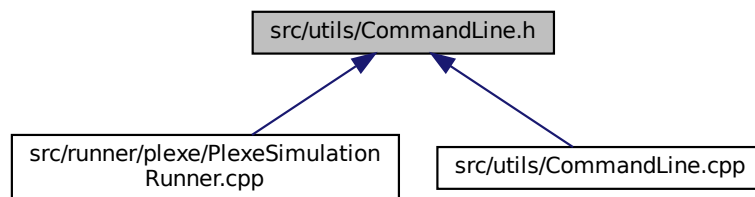
```
#include <string>
```

```
#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.

9.56.1 Detailed Description

In this file, the header of the [CommandLine](#) class is defined.

9.57 src/utils/Multithreaded.h File Reference

In this file, the header of the [Multithreaded](#) class is defined.

```
#include <mutex>
```

```
#include <queue>
```

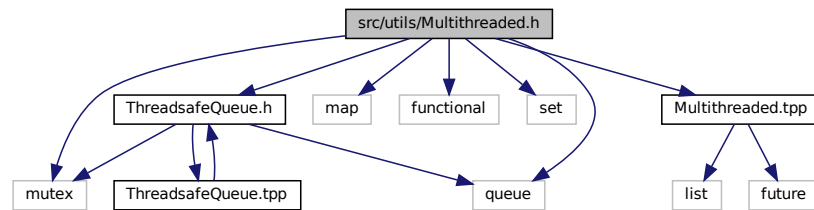
```
#include <map>
```

```
#include <functional>
```

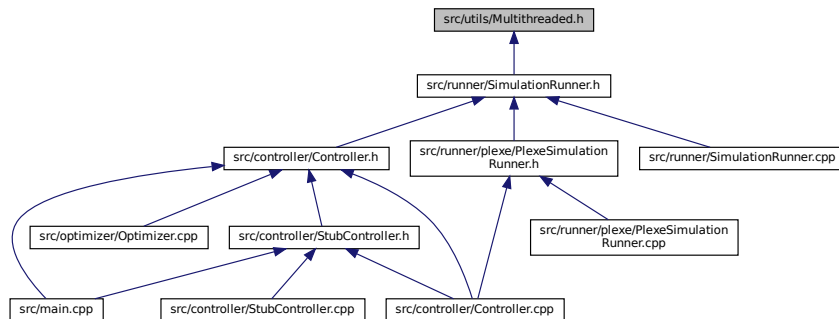
```
#include <set>
```

```
#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.

9.57.1 Detailed Description

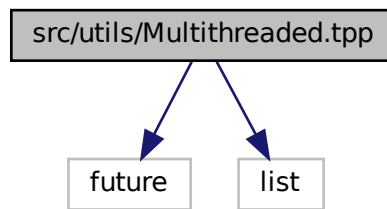
In this file, the header of the [Multithreaded](#) class is defined.

9.58 src/Utils/Multithreaded.hpp File Reference

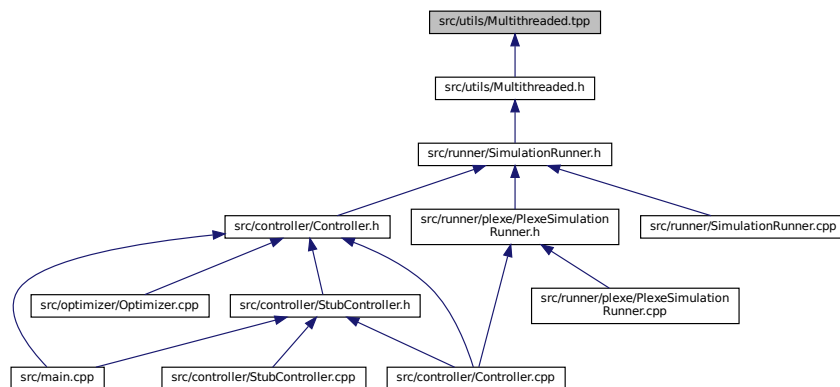
In this file, the implementation of the [Multithreaded](#) class is defined.

```
#include <future>
#include <list>
```

Include dependency graph for Multithreaded.hpp:



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



9.58.1 Detailed Description

In this file, the implementation of the [Multithreaded](#) class is defined.

9.59 src/utis/PythonScript.cpp File Reference

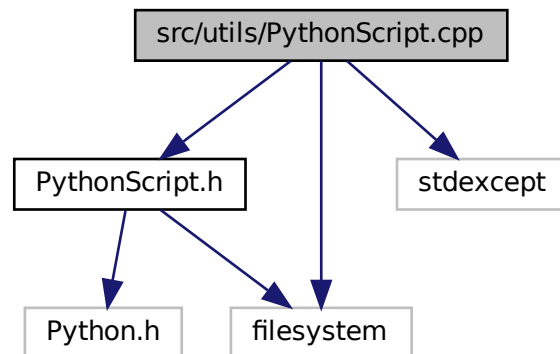
In this file, the implementation of the [PythonScript](#) class is defined.

```

#include "PythonScript.h"
#include <stdexcept>
#include <filesystem>

```

Include dependency graph for PythonScript.cpp:



9.59.1 Detailed Description

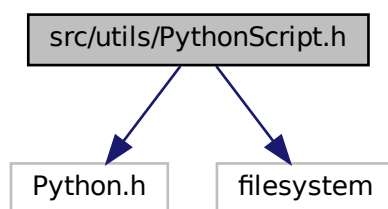
In this file, the implementation of the [PythonScript](#) class is defined.

9.60 `src/Utils/PythonScript.h` File Reference

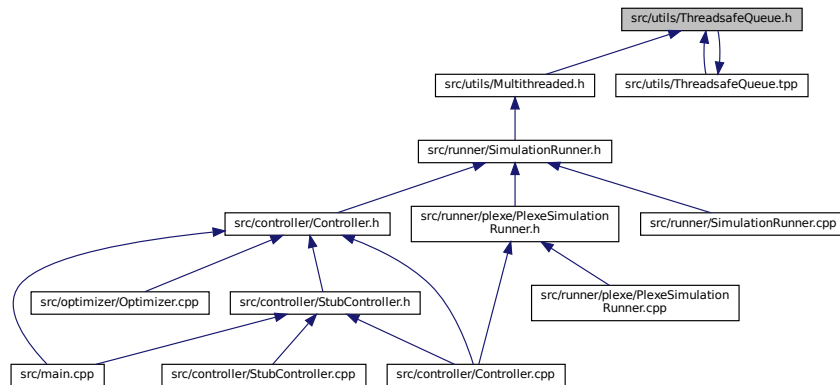
In this file, the header of the [PythonScript](#) class is defined.

```
#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.

9.61.1 Detailed Description

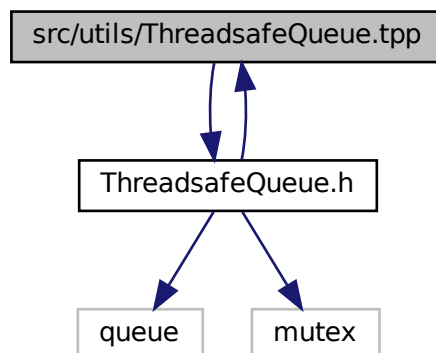
In this file, the header of the ThreadSafeQueue class is defined.

9.62 src/Utils/ThreadSafeQueue.tpp File Reference

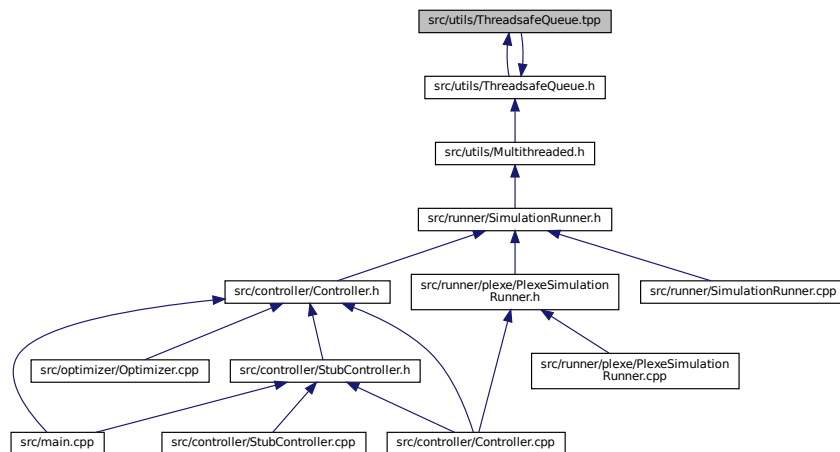
In this file, the implementation of the ThreadSafeQueue class is defined.

```
#include "ThreadSafeQueue.h"
```

Include dependency graph for ThreadSafeQueue.tpp:



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



9.62.1 Detailed Description

In this file, the implementation of the `ThreadSafeQueue` class is defined.

Index

- ~PythonScript
 - PythonScript, [105](#)
- abort
 - Abortable, [28](#)
 - Controller, [52](#)
- Abortable, [27](#)
 - abort, [28](#)
 - aborted, [28](#)
- aborted
 - Abortable, [28](#)
- ACCURACY
 - StoppingCondition, [120](#)
- activeRects
 - DirectOptimizer, [61](#)
- addActiveRects
 - DirectOptimizer, [58](#)
- addValue
 - ValueMap, [130](#)
- avgValue
 - HyRect, [76](#)
- BaseRect, [28](#)
 - BaseRect, [29](#)
 - getSamplingVertices, [29](#)
- bestVal
 - StoppingCondition, [120](#)
- ChildRect, [30](#)
 - ChildRect, [31](#)
 - getSamplingVertices, [31](#)
 - operator==, [31](#)
 - parent, [32](#)
- CmpPairVectorSharedParameterFunctionvalue, [32](#)
 - operator(), [33](#)
- CmpPtrFunctionvalue, [33](#)
 - operator(), [33](#)
- CmpSharedHyrect, [34](#)
 - operator(), [34](#)
- CmpVectorSharedParameter, [35](#)
 - operator(), [35](#)
- CommandLine, [36](#)
 - exec, [36](#)
- CONFIG
 - ConfigEditor, [41](#)
- config
 - ParameterDefinition, [96](#)
- ConfigEditor, [37](#)
 - CONFIG, [41](#)
 - ConfigEditor, [38](#)
- CONTROLLER, [41](#)
 - createConfig, [38](#)
 - deleteConfig, [38](#)
 - DIR, [41](#)
 - getConfigPath, [39](#)
 - getControllerOption, [39](#)
 - getDir, [39](#)
 - getResultPath, [39](#)
 - replaceOption, [40](#)
 - RESULTS, [41](#)
 - setResultFiles, [40](#)
- constant_headway, [21](#)
- constant_headway.py
 - get_constant_headway, [142](#)
 - multithreaded, [142](#)
- ConstantHeadway, [41](#)
 - ConstantHeadway, [43](#)
 - getName, [43](#)
 - getStatus, [43](#)
 - getStatusBar, [44](#)
 - NR_THREADS, [45](#)
 - processOutput, [44](#)
 - secureValue, [45](#)
 - usedThreads, [45](#)
- ContinuousParameter, [45](#)
 - ContinuousParameter, [47](#)
 - getVal, [47](#)
 - setVal, [47](#)
 - val, [49](#)
- CONTROLLER
 - ConfigEditor, [41](#)
- Controller, [49](#)
 - abort, [52](#)
 - Controller, [51](#)
 - evaluate, [52](#)
 - evaluation, [54](#)
 - getValueMap, [52](#)
 - keepFiles, [54](#)
 - optimizer, [54](#)
 - printValues, [54](#)
 - removeOldResultfiles, [52](#)
 - requestValues, [52](#)
 - run, [53](#)
 - runner, [54](#)
 - runSimulations, [53](#)
 - saveValues, [53](#)
 - statusBar, [54](#)
 - statusInterval, [55](#)
 - topResults, [55](#)

- updateStatus, 54
 - valueMap, 55
- controller, 19
 - Optimizer, 88
 - stepState, 19
- Controller.cpp
 - getConfigByPath, 136
- Controller::stepstate, 115
 - currentStep, 116
 - get, 116
 - next, 116
 - stepChanged, 116
- coordinate
 - Types.h, 180
- createConfig
 - ConfigEditor, 38
- ctr
 - main.cpp, 147
- currentLevel
 - Levels, 80
- currentStep
 - Controller::stepstate, 116
- D
 - DirectOptimizer, 61
 - HyRect, 76
- definition
 - Parameter, 94
- deleteConfig
 - ConfigEditor, 38
- denormalize
 - ParameterNormalizer, 98
- depth
 - DirectTypes.h, 151
- dimension
 - DirectTypes.h, 151
- DIR
 - ConfigEditor, 41
- dirCoordinate
 - DirectTypes.h, 151
- direct, 19
 - level, 20
- DirectOptimizer, 55
 - activeRects, 61
 - addActiveRects, 58
 - D, 61
 - DirectOptimizer, 57
 - estimatedValue, 58
 - getName, 58
 - getPartitionSize, 59
 - getStatus, 59
 - getStatusBar, 59
 - getValues, 59
 - iterations, 61
 - level, 61
 - normalizer, 61
 - optimalRectangles, 60
 - removeActiveRects, 60
 - runOptimization, 60
 - saveProgress, 60
 - stopCon, 62
 - trackProgress, 62
- DirectTypes.h
 - depth, 151
 - dimension, 151
 - dirCoordinate, 151
- DiscreteParameter, 62
 - DiscreteParameter, 63, 64
 - getOffset, 64
 - getStep, 64
 - getTimes, 64
 - getVal, 64
 - offset, 65
 - setTimes, 65
 - setVal, 65
 - step, 65
 - times, 66
- divide
 - HyRect, 72
- editor
 - PlexeSimulationRunner, 103
- END_TIME
 - StoppingCondition, 120
- estimatedValue
 - DirectOptimizer, 58
- evaluate
 - Controller, 52
 - StoppingCondition, 118
 - StubController, 123
- Evaluation, 66
 - getName, 67
 - getStatus, 67
 - getStatusBar, 68
 - processOutput, 68
- evaluation, 22
 - Controller, 54
- exec
 - CommandLine, 36
- f
 - StubController, 124
- functions
 - StubController, 124
- functionValue
 - Types.h, 180
- get
 - Controller::stepstate, 116
- get_constant_headway
 - constant_headway.py, 142
- getAvgValue
 - HyRect, 72
- getConditionFromJSON
 - StoppingCondition.cpp, 162
- getConfig
 - Parameter, 90
 - ParameterDefinition, 95

- getConfigByPath
 - Controller.cpp, 136
- getConfigPath
 - ConfigEditor, 39
- getControllerOption
 - ConfigEditor, 39
- getD
 - HyRect, 72
- getDepth
 - HyRect, 73
- getDiagonalLength
 - HyRect, 73
- getDir
 - ConfigEditor, 39
- getEpsilon
 - Levels, 78
- getIterationsSinceImprov
 - StoppingCondition, 119
- getLevel
 - Levels, 78
- getMax
 - Parameter, 91
 - ParameterDefinition, 96
- getMedian
 - ValueMap, 130
- getMin
 - Parameter, 91
 - ParameterDefinition, 96
- getName
 - ConstantHeadway, 43
 - DirectOptimizer, 58
 - Evaluation, 67
 - Optimizer, 87
 - PlexeSimulationRunner, 101
 - SimulationRunner, 108
 - Status, 110
- getOffset
 - DiscreteParameter, 64
- getPartitionSize
 - DirectOptimizer, 59
- getPos
 - HyRect, 73
- getRectSubset
 - Levels, 78
- getResultPath
 - ConfigEditor, 39
- getRunId
 - PlexeSimulationRunner, 102
- getSamplingVertices
 - BaseRect, 29
 - ChildRect, 31
 - HyRect, 73
- getSize
 - ThreadsafeQueue< Key >, 125
 - ValueMap, 130
- getSplitDim
 - HyRect, 73
- getStartSize
 - ThreadsafeQueue< Key >, 126
- getStatus
 - ConstantHeadway, 43
 - DirectOptimizer, 59
 - Evaluation, 67
 - Optimizer, 87
 - PlexeSimulationRunner, 102
 - SimulationRunner, 108
 - Status, 111
- getStatusBar
 - ConstantHeadway, 44
 - DirectOptimizer, 59
 - Evaluation, 68
 - Optimizer, 87
 - PlexeSimulationRunner, 102
 - SimulationRunner, 108
 - Status, 111
- getStep
 - DiscreteParameter, 64
- getTimes
 - DiscreteParameter, 64
- getTopVals
 - ValueMap, 130
- getUnit
 - Parameter, 91
 - ParameterDefinition, 96
- getVal
 - ContinuousParameter, 47
 - DiscreteParameter, 64
 - Parameter, 91
- getValueMap
 - Controller, 52
 - Optimizer, 88
- getValues
 - DirectOptimizer, 59
 - ValueMap, 131
- global
 - Levels, 80
- GrahamScan, 69
 - scan, 69
- hartman
 - StubController.cpp, 138
- HyRect, 70
 - avgValue, 76
 - D, 76
 - divide, 72
 - getAvgValue, 72
 - getD, 72
 - getDepth, 73
 - getDiagonalLength, 73
 - getPos, 73
 - getSamplingVertices, 73
 - getSplitDim, 73
 - HyRect, 71
 - operator!=, 74
 - operator<, 74
 - operator<=, 74
 - operator>, 75

- operator>=, 75
- operator==, 75
- pos, 76
- setAvgValue, 76
- t, 76
- hyrect, 24
 - position, 25
- insert
 - ValueMap, 131
- interruptHandler
 - main.cpp, 147
- isGlobal
 - Levels, 79
- isKnown
 - ValueMap, 131
- isTopValue
 - ValueMap, 132
- iterations
 - DirectOptimizer, 61
- iterationsSinceImprov
 - StoppingCondition, 120
- keepFiles
 - Controller, 54
- L0_EPSILON
 - Levels, 80
- L0_SIZE
 - Levels, 80
- L1_EPSILON
 - Levels, 80
- L1_SIZE
 - Levels, 80
- L2_EPSILON
 - Levels, 80
- L2_SIZE
 - Levels, 81
- L3_EPSILON
 - Levels, 81
- L3_SIZE
 - Levels, 81
- LARGE_DIVIDER
 - StatusBar, 114
- lastStatus
 - StatusBar, 114
- lastStep
 - StatusBar, 115
- lastVal
 - StatusBar, 115
- level
 - direct, 20
 - DirectOptimizer, 61
- Levels, 77
 - currentLevel, 80
 - getEpsilon, 78
 - getLevel, 78
 - getRectSubset, 78
 - global, 80
 - isGlobal, 79
 - L0_EPSILON, 80
 - L0_SIZE, 80
 - L1_EPSILON, 80
 - L1_SIZE, 80
 - L2_EPSILON, 80
 - L2_SIZE, 81
 - L3_EPSILON, 81
 - L3_SIZE, 81
 - nextLevel, 79
 - setGlobal, 79
- lowerValues
 - ValueMap, 132
- main
 - main.cpp, 147
- main.cpp
 - ctr, 147
 - interruptHandler, 147
 - main, 147
- max
 - ParameterDefinition, 96
- min
 - ParameterDefinition, 97
- mins
 - StoppingCondition, 120
- Multithreaded
 - Multithreaded< Key, T, Compare, Allocator >, 83
- multithreaded
 - constant_headway.py, 142
- Multithreaded< Key, T, Compare, Allocator >, 81
 - Multithreaded, 83
 - multithreadFunction, 83
 - NR_THREADS, 84
 - queue, 85
 - runMultithreadedFunctions, 84
 - work, 84
- multithreadFunction
 - Multithreaded< Key, T, Compare, Allocator >, 83
- next
 - Controller::stepstate, 116
- nextLevel
 - Levels, 79
- NO_NAME
 - Status, 111
- NO_STATUS_SUPPORT
 - Status, 111
- normalize
 - ParameterNormalizer, 98
- normalizer
 - DirectOptimizer, 61
- NR_ACCURACY_ITERATIONS
 - StoppingCondition, 120
- NR_EVALUATIONS
 - StoppingCondition, 120
- NR_HYRECTS
 - StoppingCondition, 121
- NR_THREADS

- ConstantHeadway, [45](#)
- Multithreaded< Key, T, Compare, Allocator >, [84](#)
- offset
 - DiscreteParameter, [65](#)
- operationsLock
 - ValueMap, [133](#)
- operator!=
 - HyRect, [74](#)
 - Parameter, [92](#)
- operator<
 - HyRect, [74](#)
 - Parameter, [92](#)
- operator<=
 - HyRect, [74](#)
 - Parameter, [92](#)
- operator>
 - HyRect, [75](#)
 - Parameter, [93](#)
- operator>=
 - HyRect, [75](#)
 - Parameter, [93](#)
- operator()
 - CmpPairVectorSharedParameterFunctionvalue, [33](#)
 - CmpPtrFunctionvalue, [33](#)
 - CmpSharedHyrect, [34](#)
 - CmpVectorSharedParameter, [35](#)
- operator==
 - ChildRect, [31](#)
 - HyRect, [75](#)
 - Parameter, [93](#)
- optimalRectangles
 - DirectOptimizer, [60](#)
- Optimizer, [85](#)
 - controller, [88](#)
 - getName, [87](#)
 - getStatus, [87](#)
 - getStatusBar, [87](#)
 - getValueMap, [88](#)
 - Optimizer, [87](#)
 - parameters, [89](#)
 - requestValues, [88](#)
 - runOptimization, [88](#)
- optimizer, [24](#)
 - Controller, [54](#)
- Parameter, [89](#)
 - definition, [94](#)
 - getConfig, [90](#)
 - getMax, [91](#)
 - getMin, [91](#)
 - getUnit, [91](#)
 - getVal, [91](#)
 - operator!=, [92](#)
 - operator<, [92](#)
 - operator<=, [92](#)
 - operator>, [93](#)
 - operator>=, [93](#)
 - operator==, [93](#)
- Parameter, [90](#)
 - setVal, [94](#)
- parameterCombination
 - Types.h, [181](#)
- ParameterDefinition, [94](#)
 - config, [96](#)
 - getConfig, [95](#)
 - getMax, [96](#)
 - getMin, [96](#)
 - getUnit, [96](#)
 - max, [96](#)
 - min, [97](#)
 - ParameterDefinition, [95](#)
 - unit, [97](#)
- ParameterNormalizer, [97](#)
 - denormalize, [98](#)
 - normalize, [98](#)
 - ParameterNormalizer, [98](#)
 - parameters, [99](#)
- parameters, [22](#)
 - Optimizer, [89](#)
 - ParameterNormalizer, [99](#)
- parent
 - ChildRect, [32](#)
- pFunc
 - PythonScript, [105](#)
- plexex, [21](#)
- PlexeSimulationRunner, [99](#)
 - editor, [103](#)
 - getName, [101](#)
 - getRunId, [102](#)
 - getStatus, [102](#)
 - getStatusBar, [102](#)
 - PlexeSimulationRunner, [101](#)
 - REPEAT, [103](#)
 - runNumber, [103](#)
 - runNumberLock, [103](#)
 - SCENARIOS, [104](#)
 - work, [102, 103](#)
- pModule
 - PythonScript, [105](#)
- pop
 - ThreadsafeQueue< Key >, [126](#)
- pos
 - HyRect, [76](#)
- position
 - hyrect, [25](#)
- printResult
 - StatusBar, [113](#)
- printResults
 - StatusBar, [113](#)
- printStatus
 - StatusBar, [113](#)
- printValues
 - Controller, [54](#)
- processOutput
 - ConstantHeadway, [44](#)
 - Evaluation, [68](#)

- push
 - ThreadsafeQueue< Key >, 126
- PythonScript, 104
 - ~PythonScript, 105
 - pFunc, 105
 - pModule, 105
 - PythonScript, 105
- query
 - ValueMap, 132
- queue
 - Multithreaded< Key, T, Compare, Allocator >, 85
- queueLock
 - ThreadsafeQueue< Key >, 126
- removeActiveRects
 - DirectOptimizer, 60
- removeOldResultfiles
 - Controller, 52
 - StubController, 123
- REPEAT
 - PlexeSimulationRunner, 103
- replaceOption
 - ConfigEditor, 40
- requestValues
 - Controller, 52
 - Optimizer, 88
- RESULTS
 - ConfigEditor, 41
- run
 - Controller, 53
- runId
 - Types.h, 181
- runMultithreadedFunctions
 - Multithreaded< Key, T, Compare, Allocator >, 84
- runner, 23
 - Controller, 54
- runNumber
 - PlexeSimulationRunner, 103
- runNumberLock
 - PlexeSimulationRunner, 103
- runOptimization
 - DirectOptimizer, 60
 - Optimizer, 88
- runSimulations
 - Controller, 53
 - SimulationRunner, 109
 - StubController, 123
- safeQueue
 - ThreadsafeQueue< Key >, 127
- saveProgress
 - DirectOptimizer, 60
- saveValues
 - Controller, 53
- scan
 - GrahamScan, 69
- SCENARIOS
 - PlexeSimulationRunner, 104
- secureValue
 - ConstantHeadway, 45
- setAvgValue
 - HyRect, 76
- setGlobal
 - Levels, 79
- setResultFiles
 - ConfigEditor, 40
- setStartNow
 - StoppingCondition, 119
- setTimes
 - DiscreteParameter, 65
- setVal
 - ContinuousParameter, 47
 - DiscreteParameter, 65
 - Parameter, 94
- shekel
 - StubController.cpp, 139
- SimulationRunner, 106
 - getName, 108
 - getStatus, 108
 - getStatusBar, 108
 - runSimulations, 109
 - SimulationRunner, 108
 - work, 109
- SMALL_DIVIDER
 - StatusBar, 115
- src/ComparisonFunctions.h, 135
- src/controller/Controller.cpp, 136
- src/controller/Controller.h, 137
- src/controller/StubController.cpp, 138
- src/controller/StubController.h, 139
- src/controller/ValueMap.cpp, 140
- src/controller/ValueMap.h, 140
- src/evaluation/constant_headway/constant_headway.py, 141
- src/evaluation/constant_headway/ConstantHeadway.cpp, 143
- src/evaluation/constant_headway/ConstantHeadway.h, 143
- src/evaluation/Evaluation.cpp, 144
- src/evaluation/Evaluation.h, 145
- src/main.cpp, 146
- src/optimizer/direct/DirectComparisonFunctions.h, 147
- src/optimizer/direct/DirectOptimizer.cpp, 148
- src/optimizer/direct/DirectOptimizer.h, 149
- src/optimizer/direct/DirectTypes.h, 150
- src/optimizer/direct/GrahamScan.cpp, 151
- src/optimizer/direct/GrahamScan.h, 151
- src/optimizer/direct/hyrect/BaseRect.cpp, 152
- src/optimizer/direct/hyrect/BaseRect.h, 153
- src/optimizer/direct/hyrect/ChildRect.cpp, 154
- src/optimizer/direct/hyrect/ChildRect.h, 155
- src/optimizer/direct/hyrect/HyRect.cpp, 155
- src/optimizer/direct/hyrect/HyRect.h, 156
- src/optimizer/direct/Levels.cpp, 157
- src/optimizer/direct/Levels.h, 158
- src/optimizer/direct/ParameterNormalizer.cpp, 159

- src/optimizer/direct/ParameterNormalizer.h, 160
- src/optimizer/direct/StoppingCondition.cpp, 161
- src/optimizer/direct/StoppingCondition.h, 162
- src/optimizer/Optimizer.cpp, 163
- src/optimizer/Optimizer.h, 164
- src/parameters/ContinuousParameter.cpp, 165
- src/parameters/ContinuousParameter.h, 165
- src/parameters/DiscreteParameter.cpp, 166
- src/parameters/DiscreteParameter.h, 167
- src/parameters/Parameter.cpp, 168
- src/parameters/Parameter.h, 169
- src/parameters/ParameterDefinition.cpp, 170
- src/parameters/ParameterDefinition.h, 171
- src/runner/plex/ConfigEditor.cpp, 172
- src/runner/plex/ConfigEditor.h, 173
- src/runner/plex/PlexSimulationRunner.cpp, 174
- src/runner/plex/PlexSimulationRunner.h, 174
- src/runner/SimulationRunner.cpp, 175
- src/runner/SimulationRunner.h, 175
- src/status/Status.cpp, 176
- src/status/Status.h, 177
- src/status/StatusBar.cpp, 178
- src/status/StatusBar.h, 178
- src/Types.h, 179
- src/Utils/Abortable.cpp, 181
- src/Utils/Abortable.h, 181
- src/Utils/CommandLine.cpp, 182
- src/Utils/CommandLine.h, 183
- src/Utils/Multithreaded.h, 183
- src/Utils/Multithreaded.tpp, 184
- src/Utils/PythonScript.cpp, 185
- src/Utils/PythonScript.h, 186
- src/Utils/ThreadsafeQueue.h, 187
- src/Utils/ThreadsafeQueue.tpp, 188
- startSize
 - ThreadsafeQueue< Key >, 127
- Status, 109
 - getName, 110
 - getStatus, 111
 - getStatusBar, 111
 - NO_NAME, 111
 - NO_STATUS_SUPPORT, 111
- status, 23
 - step, 24
- StatusBar, 112
 - LARGE_DIVIDER, 114
 - lastStatus, 114
 - lastStep, 115
 - lastVal, 115
 - printResult, 113
 - printResults, 113
 - printStatus, 113
 - SMALL_DIVIDER, 115
 - updateStatus, 114
- statusBar
 - Controller, 54
- statusInterval
 - Controller, 55
- step
 - DiscreteParameter, 65
 - status, 24
- stepChanged
 - Controller::stepstate, 116
- stepState
 - controller, 19
- stopCon
 - DirectOptimizer, 62
- StoppingCondition, 117
 - ACCURACY, 120
 - bestVal, 120
 - END_TIME, 120
 - evaluate, 118
 - getIterationsSinceImprov, 119
 - iterationsSinceImprov, 120
 - mins, 120
 - NR_ACCURACY_ITERATIONS, 120
 - NR_EVALUATIONS, 120
 - NR_HYRECTS, 121
 - setStartNow, 119
 - StoppingCondition, 118
 - time_eval, 121
 - updateAccuracy, 119
- StoppingCondition.cpp
 - getConditionFromJSON, 162
- StubController, 121
 - evaluate, 123
 - f, 124
 - functions, 124
 - removeOldResultfiles, 123
 - runSimulations, 123
 - StubController, 123
 - updateStatus, 124
- StubController.cpp
 - hartman, 138
 - shekel, 139
- t
 - HyRect, 76
- tba
 - ValueMap, 133
- ThreadsafeQueue< Key >, 125
 - getSize, 125
 - getStartSize, 126
 - pop, 126
 - push, 126
 - queueLock, 126
 - safeQueue, 127
 - startSize, 127
- time_eval
 - StoppingCondition, 121
- times
 - DiscreteParameter, 66
- topEntries
 - ValueMap, 133
- topResults
 - Controller, 55
- topVals

- ValueMap, [133](#)
- trackProgress
 - DirectOptimizer, [62](#)
- Types.h
 - coordinate, [180](#)
 - functionValue, [180](#)
 - parameterCombination, [181](#)
 - runId, [181](#)
- unit
 - ParameterDefinition, [97](#)
- updateAccuracy
 - StoppingCondition, [119](#)
- updateMap
 - ValueMap, [132](#)
- updateStatus
 - Controller, [54](#)
 - StatusBar, [114](#)
 - StubController, [124](#)
- upperValues
 - ValueMap, [133](#)
- usedThreads
 - ConstantHeadway, [45](#)
- utils, [25](#)
- val
 - ContinuousParameter, [49](#)
- ValueMap, [127](#)
 - addValue, [130](#)
 - getMedian, [130](#)
 - getSize, [130](#)
 - getTopVals, [130](#)
 - getValues, [131](#)
 - insert, [131](#)
 - isKnown, [131](#)
 - isTopValue, [132](#)
 - lowerValues, [132](#)
 - operationsLock, [133](#)
 - query, [132](#)
 - tba, [133](#)
 - topEntries, [133](#)
 - topVals, [133](#)
 - updateMap, [132](#)
 - upperValues, [133](#)
 - ValueMap, [128](#)
 - values, [133](#)
- valueMap
 - Controller, [55](#)
- values
 - ValueMap, [133](#)
- work
 - Multithreaded< Key, T, Compare, Allocator >, [84](#)
 - PlexeSimulationRunner, [102](#), [103](#)
 - SimulationRunner, [109](#)