

MPCOTool

4.10.2

Generated by Doxygen 1.9.8



<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 Experiment Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 name	5
3.1.2.2 ninputs	6
3.1.2.3 stencil	6
3.1.2.4 template_flags	6
3.1.2.5 weight	6
3.2 Input Struct Reference	6
3.2.1 Detailed Description	8
3.2.2 Field Documentation	8
3.2.2.1 adaptation_ratio	8
3.2.2.2 algorithm	8
3.2.2.3 cleaner	8
3.2.2.4 climbing	9
3.2.2.5 directory	9
3.2.2.6 evaluator	9
3.2.2.7 experiment	9
3.2.2.8 mutation_ratio	9
3.2.2.9 name	9
3.2.2.10 nbest	10
3.2.2.11 nestimates	10
3.2.2.12 nexperiments	10
3.2.2.13 nfinal_steps	10
3.2.2.14 niterations	10
3.2.2.15 norm	10
3.2.2.16 nsimulations	11
3.2.2.17 nsteps	11
3.2.2.18 nvariables	11
3.2.2.19 p	11
3.2.2.20 relaxation	11
3.2.2.21 reproduction_ratio	11
3.2.2.22 result	12
3.2.2.23 seed	12
3.2.2.24 simulator	12
3.2.2.25 template_flags	12

3.2.2.26 threshold	12
3.2.2.27 tolerance	12
3.2.2.28 type	13
3.2.2.29 variable	13
3.2.2.30 variables	13
3.3 Optimize Struct Reference	13
3.3.1 Detailed Description	16
3.3.2 Field Documentation	16
3.3.2.1 adaptation_ratio	16
3.3.2.2 algorithm	16
3.3.2.3 calculation_time	16
3.3.2.4 cleaner	17
3.3.2.5 climbing	17
3.3.2.6 error_best	17
3.3.2.7 error_old	17
3.3.2.8 evaluator	17
3.3.2.9 experiment	17
3.3.2.10 file	18
3.3.2.11 file_result	18
3.3.2.12 file_variables	18
3.3.2.13 genetic_variable	18
3.3.2.14 label	18
3.3.2.15 mpi_rank	18
3.3.2.16 mutation_ratio	19
3.3.2.17 nbest	19
3.3.2.18 nbits	19
3.3.2.19 nend	19
3.3.2.20 nend_climbing	19
3.3.2.21 nestimates	19
3.3.2.22 nexperiments	20
3.3.2.23 nfinal_steps	20
3.3.2.24 ninputs	20
3.3.2.25 niterations	20
3.3.2.26 nsaveds	20
3.3.2.27 nsimulations	20
3.3.2.28 nstart	21
3.3.2.29 nstart_climbing	21
3.3.2.30 nsteps	21
3.3.2.31 nsweeps	21
3.3.2.32 nvariables	21
3.3.2.33 p	21
3.3.2.34 precision	22

3.3.2.35 rangemax	22
3.3.2.36 rangemaxabs	22
3.3.2.37 rangemin	22
3.3.2.38 rangeminabs	22
3.3.2.39 relaxation	22
3.3.2.40 reproduction_ratio	23
3.3.2.41 result	23
3.3.2.42 rng	23
3.3.2.43 seed	23
3.3.2.44 simulation_best	23
3.3.2.45 simulator	23
3.3.2.46 step	24
3.3.2.47 stop	24
3.3.2.48 template_flags	24
3.3.2.49 thread	24
3.3.2.50 thread_climbing	24
3.3.2.51 threshold	24
3.3.2.52 tolerance	25
3.3.2.53 value	25
3.3.2.54 value_old	25
3.3.2.55 variables	25
3.3.2.56 weight	25
3.4 Options Struct Reference	25
3.4.1 Detailed Description	26
3.4.2 Field Documentation	26
3.4.2.1 dialog	26
3.4.2.2 grid	26
3.4.2.3 label_climbing	26
3.4.2.4 label_seed	27
3.4.2.5 label_threads	27
3.4.2.6 spin_climbing	27
3.4.2.7 spin_seed	27
3.4.2.8 spin_threads	27
3.5 ParallelData Struct Reference	27
3.5.1 Detailed Description	28
3.5.2 Field Documentation	28
3.5.2.1 thread	28
3.6 Running Struct Reference	28
3.6.1 Detailed Description	28
3.6.2 Field Documentation	29
3.6.2.1 dialog	29
3.6.2.2 grid	29

---

3.6.2.3 label . . . . .	29
3.6.2.4 spinner . . . . .	29
3.7 Variable Struct Reference . . . . .	29
3.7.1 Detailed Description . . . . .	30
3.7.2 Field Documentation . . . . .	30
3.7.2.1 name . . . . .	30
3.7.2.2 nbits . . . . .	30
3.7.2.3 nsweeps . . . . .	31
3.7.2.4 precision . . . . .	31
3.7.2.5 rangemax . . . . .	31
3.7.2.6 rangemaxabs . . . . .	31
3.7.2.7 rangemin . . . . .	31
3.7.2.8 rangeminabs . . . . .	31
3.7.2.9 step . . . . .	32
3.8 Window Struct Reference . . . . .	32
3.8.1 Detailed Description . . . . .	37
3.8.2 Field Documentation . . . . .	37
3.8.2.1 application_directory . . . . .	37
3.8.2.2 box_buttons . . . . .	37
3.8.2.3 button_about . . . . .	37
3.8.2.4 button_add_experiment . . . . .	37
3.8.2.5 button_add_variable . . . . .	38
3.8.2.6 button_algorithm . . . . .	38
3.8.2.7 button_cleaner . . . . .	38
3.8.2.8 button_climbing . . . . .	38
3.8.2.9 button_evaluator . . . . .	38
3.8.2.10 button_exit . . . . .	38
3.8.2.11 button_experiment . . . . .	39
3.8.2.12 button_help . . . . .	39
3.8.2.13 button_norm . . . . .	39
3.8.2.14 button_open . . . . .	39
3.8.2.15 button_options . . . . .	39
3.8.2.16 button_remove_experiment . . . . .	39
3.8.2.17 button_remove_variable . . . . .	40
3.8.2.18 button_run . . . . .	40
3.8.2.19 button_save . . . . .	40
3.8.2.20 button_simulator . . . . .	40
3.8.2.21 button_template . . . . .	40
3.8.2.22 check_cleaner . . . . .	40
3.8.2.23 check_climbing . . . . .	41
3.8.2.24 check_evaluator . . . . .	41
3.8.2.25 check_maxabs . . . . .	41

---

3.8.2.26	check_minabs	41
3.8.2.27	check_template	41
3.8.2.28	combo_experiment	41
3.8.2.29	combo_variable	42
3.8.2.30	entry_result	42
3.8.2.31	entry_variable	42
3.8.2.32	entry_variables	42
3.8.2.33	experiment	42
3.8.2.34	frame_algorithm	42
3.8.2.35	frame_experiment	43
3.8.2.36	frame_norm	43
3.8.2.37	frame_variable	43
3.8.2.38	grid	43
3.8.2.39	grid_algorithm	43
3.8.2.40	grid_climbing	43
3.8.2.41	grid_experiment	44
3.8.2.42	grid_files	44
3.8.2.43	grid_norm	44
3.8.2.44	grid_variable	44
3.8.2.45	id_experiment	44
3.8.2.46	id_experiment_name	44
3.8.2.47	id_input	45
3.8.2.48	id_template	45
3.8.2.49	id_variable	45
3.8.2.50	id_variable_label	45
3.8.2.51	label_adaptation	45
3.8.2.52	label_bests	45
3.8.2.53	label_bits	46
3.8.2.54	label_estimates	46
3.8.2.55	label_experiment	46
3.8.2.56	label_final_steps	46
3.8.2.57	label_generations	46
3.8.2.58	label_iterations	46
3.8.2.59	label_max	47
3.8.2.60	label_min	47
3.8.2.61	label_mutation	47
3.8.2.62	label_p	47
3.8.2.63	label_population	47
3.8.2.64	label_precision	47
3.8.2.65	label_relaxation	48
3.8.2.66	label_reproduction	48
3.8.2.67	label_result	48

3.8.2.68 label_simulations	48
3.8.2.69 label_simulator	48
3.8.2.70 label_step	48
3.8.2.71 label_steps	49
3.8.2.72 label_sweeps	49
3.8.2.73 label_threshold	49
3.8.2.74 label_tolerance	49
3.8.2.75 label_variable	49
3.8.2.76 label_variables	49
3.8.2.77 label_weight	50
3.8.2.78 logo	50
3.8.2.79 nexperiments	50
3.8.2.80 nvariables	50
3.8.2.81 scrolled_max	50
3.8.2.82 scrolled_maxabs	50
3.8.2.83 scrolled_min	51
3.8.2.84 scrolled_minabs	51
3.8.2.85 scrolled_p	51
3.8.2.86 scrolled_step	51
3.8.2.87 scrolled_threshold	51
3.8.2.88 spin_adaptation	51
3.8.2.89 spin_bests	52
3.8.2.90 spin_bits	52
3.8.2.91 spin_estimates	52
3.8.2.92 spin_final_steps	52
3.8.2.93 spin_generations	52
3.8.2.94 spin_iterations	52
3.8.2.95 spin_max	53
3.8.2.96 spin_maxabs	53
3.8.2.97 spin_min	53
3.8.2.98 spin_minabs	53
3.8.2.99 spin_mutation	53
3.8.2.100 spin_p	53
3.8.2.101 spin_population	54
3.8.2.102 spin_precision	54
3.8.2.103 spin_relaxation	54
3.8.2.104 spin_reproduction	54
3.8.2.105 spin_simulations	54
3.8.2.106 spin_step	54
3.8.2.107 spin_steps	55
3.8.2.108 spin_sweeps	55
3.8.2.109 spin_threshold	55



3.8.2.110 spin_tolerance . . . . .	55
3.8.2.111 spin_weight . . . . .	55
3.8.2.112 variable . . . . .	55
3.8.2.113 window . . . . .	55
<b>4 File Documentation</b>	<b>57</b>
4.1 config.h File Reference . . . . .	57
4.1.1 Detailed Description . . . . .	60
4.1.2 Macro Definition Documentation . . . . .	61
4.1.2.1 DEFAULT_PRECISION . . . . .	61
4.1.2.2 DEFAULT_RANDOM_SEED . . . . .	61
4.1.2.3 DEFAULT_RELAXATION . . . . .	61
4.1.2.4 LABEL_ABSOLUTE_MAXIMUM . . . . .	61
4.1.2.5 LABEL_ABSOLUTE_MINIMUM . . . . .	61
4.1.2.6 LABEL_ADAPTATION . . . . .	61
4.1.2.7 LABEL_ALGORITHM . . . . .	62
4.1.2.8 LABEL_CLEANER . . . . .	62
4.1.2.9 LABEL_CLIMBING . . . . .	62
4.1.2.10 LABEL_COORDINATES . . . . .	62
4.1.2.11 LABEL_EUCLIDIAN . . . . .	62
4.1.2.12 LABEL_EVALUATOR . . . . .	62
4.1.2.13 LABEL_EXPERIMENT . . . . .	63
4.1.2.14 LABEL_EXPERIMENTS . . . . .	63
4.1.2.15 LABEL_GENETIC . . . . .	63
4.1.2.16 LABEL_INPUT1 . . . . .	63
4.1.2.17 LABEL_INPUT2 . . . . .	63
4.1.2.18 LABEL_INPUT3 . . . . .	63
4.1.2.19 LABEL_INPUT4 . . . . .	64
4.1.2.20 LABEL_INPUT5 . . . . .	64
4.1.2.21 LABEL_INPUT6 . . . . .	64
4.1.2.22 LABEL_INPUT7 . . . . .	64
4.1.2.23 LABEL_INPUT8 . . . . .	64
4.1.2.24 LABEL_MAXIMUM . . . . .	64
4.1.2.25 LABEL_MINIMUM . . . . .	65
4.1.2.26 LABEL_MONTE_CARLO . . . . .	65
4.1.2.27 LABEL_MUTATION . . . . .	65
4.1.2.28 LABEL_NAME . . . . .	65
4.1.2.29 LABEL_NBEST . . . . .	65
4.1.2.30 LABEL_NBITS . . . . .	65
4.1.2.31 LABEL_NESTIMATES . . . . .	66
4.1.2.32 LABEL_NFINAL_STEPS . . . . .	66
4.1.2.33 LABEL_NGENERATIONS . . . . .	66

4.1.2.34 LABEL_NITERATIONS . . . . .	66
4.1.2.35 LABEL_NORM . . . . .	66
4.1.2.36 LABEL_NPOPULATION . . . . .	66
4.1.2.37 LABEL_NSIMULATIONS . . . . .	67
4.1.2.38 LABEL_NSTEPS . . . . .	67
4.1.2.39 LABEL_NSWEEPS . . . . .	67
4.1.2.40 LABEL_OPTIMIZE . . . . .	67
4.1.2.41 LABEL_ORTHOGONAL . . . . .	67
4.1.2.42 LABEL_P . . . . .	67
4.1.2.43 LABEL_PRECISION . . . . .	68
4.1.2.44 LABEL_RANDOM . . . . .	68
4.1.2.45 LABEL_RELAXATION . . . . .	68
4.1.2.46 LABEL_REPRODUCTION . . . . .	68
4.1.2.47 LABEL_RESULT_FILE . . . . .	68
4.1.2.48 LABEL_SEED . . . . .	68
4.1.2.49 LABEL_SIMULATOR . . . . .	69
4.1.2.50 LABEL_STEP . . . . .	69
4.1.2.51 LABEL_SWEEP . . . . .	69
4.1.2.52 LABEL_TAXICAB . . . . .	69
4.1.2.53 LABEL_TEMPLATE1 . . . . .	69
4.1.2.54 LABEL_TEMPLATE2 . . . . .	69
4.1.2.55 LABEL_TEMPLATE3 . . . . .	70
4.1.2.56 LABEL_TEMPLATE4 . . . . .	70
4.1.2.57 LABEL_TEMPLATE5 . . . . .	70
4.1.2.58 LABEL_TEMPLATE6 . . . . .	70
4.1.2.59 LABEL_TEMPLATE7 . . . . .	70
4.1.2.60 LABEL_TEMPLATE8 . . . . .	70
4.1.2.61 LABEL_THRESHOLD . . . . .	71
4.1.2.62 LABEL_TOLERANCE . . . . .	71
4.1.2.63 LABEL_VARIABLE . . . . .	71
4.1.2.64 LABEL_VARIABLES . . . . .	71
4.1.2.65 LABEL_VARIABLES_FILE . . . . .	71
4.1.2.66 LABEL_WEIGHT . . . . .	71
4.1.2.67 LOCALE_DIR . . . . .	72
4.1.2.68 MAX_NINPUTS . . . . .	72
4.1.2.69 NALGORITHMS . . . . .	72
4.1.2.70 NCLIMBINGS . . . . .	72
4.1.2.71 NNORMS . . . . .	72
4.1.2.72 NPRECISIONS . . . . .	72
4.1.2.73 PROGRAM_INTERFACE . . . . .	72
4.1.3 Enumeration Type Documentation . . . . .	72
4.1.3.1 INPUT_TYPE . . . . .	72

4.2 config.h . . . . .	73
4.3 experiment.c File Reference . . . . .	74
4.3.1 Detailed Description . . . . .	75
4.3.2 Macro Definition Documentation . . . . .	76
4.3.2.1 DEBUG_EXPERIMENT . . . . .	76
4.3.3 Function Documentation . . . . .	76
4.3.3.1 experiment_error() . . . . .	76
4.3.3.2 experiment_free() . . . . .	76
4.3.3.3 experiment_new() . . . . .	77
4.3.3.4 experiment_open_json() . . . . .	77
4.3.3.5 experiment_open_xml() . . . . .	79
4.3.4 Variable Documentation . . . . .	82
4.3.4.1 stencil . . . . .	82
4.3.4.2 stencilbin . . . . .	82
4.4 experiment.c . . . . .	83
4.5 experiment.h File Reference . . . . .	87
4.5.1 Detailed Description . . . . .	88
4.5.2 Function Documentation . . . . .	88
4.5.2.1 experiment_error() . . . . .	88
4.5.2.2 experiment_free() . . . . .	88
4.5.2.3 experiment_open_json() . . . . .	90
4.5.2.4 experiment_open_xml() . . . . .	92
4.5.3 Variable Documentation . . . . .	95
4.5.3.1 stencil . . . . .	95
4.5.3.2 stencilbin . . . . .	95
4.6 experiment.h . . . . .	95
4.7 input.c File Reference . . . . .	96
4.7.1 Detailed Description . . . . .	97
4.7.2 Macro Definition Documentation . . . . .	97
4.7.2.1 DEBUG_INPUT . . . . .	97
4.7.3 Function Documentation . . . . .	97
4.7.3.1 input_error() . . . . .	97
4.7.3.2 input_free() . . . . .	98
4.7.3.3 input_new() . . . . .	99
4.7.3.4 input_open() . . . . .	99
4.7.3.5 input_open_json() . . . . .	100
4.7.3.6 input_open_xml() . . . . .	106
4.7.4 Variable Documentation . . . . .	112
4.7.4.1 input . . . . .	112
4.7.4.2 result_name . . . . .	112
4.7.4.3 variables_name . . . . .	113
4.8 input.c . . . . .	113

4.9 input.h File Reference	125
4.9.1 Detailed Description	126
4.9.2 Enumeration Type Documentation	126
4.9.2.1 ClimbingMethod	126
4.9.2.2 ErrorNorm	126
4.9.3 Function Documentation	127
4.9.3.1 input_free()	127
4.9.3.2 input_new()	128
4.9.3.3 input_open()	128
4.9.4 Variable Documentation	130
4.9.4.1 input	130
4.9.4.2 result_name	130
4.9.4.3 variables_name	131
4.10 input.h	131
4.11 interface.c File Reference	132
4.11.1 Detailed Description	134
4.11.2 Macro Definition Documentation	134
4.11.2.1 DEBUG_INTERFACE	134
4.11.2.2 INPUT_FILE	134
4.11.3 Function Documentation	135
4.11.3.1 dialog_cleaner()	135
4.11.3.2 dialog_cleaner_close()	135
4.11.3.3 dialog_evaluator()	136
4.11.3.4 dialog_evaluator_close()	136
4.11.3.5 dialog_name_experiment_close()	137
4.11.3.6 dialog_open_close()	138
4.11.3.7 dialog_options_close()	139
4.11.3.8 dialog_save_close()	140
4.11.3.9 dialog_simulator()	141
4.11.3.10 dialog_simulator_close()	142
4.11.3.11 input_save()	143
4.11.3.12 input_save_climbing_json()	144
4.11.3.13 input_save_climbing_xml()	144
4.11.3.14 input_save_json()	145
4.11.3.15 input_save_xml()	148
4.11.3.16 options_new()	150
4.11.3.17 running_new()	151
4.11.3.18 window_about()	152
4.11.3.19 window_add_experiment()	152
4.11.3.20 window_add_variable()	153
4.11.3.21 window_get_algorithm()	154
4.11.3.22 window_get_climbing()	155

4.11.3.23 window_get_norm()	155
4.11.3.24 window_help()	156
4.11.3.25 window_inputs_experiment()	156
4.11.3.26 window_label_variable()	157
4.11.3.27 window_name_experiment()	157
4.11.3.28 window_new()	158
4.11.3.29 window_open()	168
4.11.3.30 window_precision_variable()	169
4.11.3.31 window_rangemax_variable()	170
4.11.3.32 window_rangemaxabs_variable()	170
4.11.3.33 window_rangemin_variable()	170
4.11.3.34 window_rangeminabs_variable()	171
4.11.3.35 window_read()	171
4.11.3.36 window_remove_experiment()	173
4.11.3.37 window_remove_variable()	174
4.11.3.38 window_run()	174
4.11.3.39 window_save()	175
4.11.3.40 window_save_climbing()	176
4.11.3.41 window_set_algorithm()	177
4.11.3.42 window_set_experiment()	178
4.11.3.43 window_set_variable()	178
4.11.3.44 window_step_variable()	179
4.11.3.45 window_template_experiment()	180
4.11.3.46 window_template_experiment_close()	181
4.11.3.47 window_update()	181
4.11.3.48 window_update_climbing()	183
4.11.3.49 window_update_variable()	184
4.11.3.50 window_weight_experiment()	185
4.11.4 Variable Documentation	185
4.11.4.1 logo	185
4.11.4.2 options	186
4.11.4.3 running	186
4.11.4.4 window	186
4.12 interface.c	186
4.13 interface.h File Reference	221
4.13.1 Detailed Description	222
4.13.2 Macro Definition Documentation	222
4.13.2.1 MAX_LENGTH	222
4.13.3 Function Documentation	222
4.13.3.1 window_new()	222
4.13.4 Variable Documentation	233
4.13.4.1 window	233

4.14 interface.h	233
4.15 main.c File Reference	236
4.15.1 Detailed Description	237
4.15.2 Macro Definition Documentation	237
4.15.2.1 JBW	237
4.15.3 Function Documentation	237
4.15.3.1 main()	237
4.16 main.c	238
4.17 mpcotool.c File Reference	239
4.17.1 Detailed Description	240
4.17.2 Macro Definition Documentation	241
4.17.2.1 DEBUG_MPCOTOOL	241
4.17.3 Function Documentation	241
4.17.3.1 mpcotool()	241
4.18 mpcotool.c	243
4.19 mpcotool.h File Reference	246
4.19.1 Detailed Description	246
4.19.2 Function Documentation	247
4.19.2.1 mpcotool()	247
4.20 mpcotool.h	249
4.21 optimize.c File Reference	250
4.21.1 Detailed Description	252
4.21.2 Macro Definition Documentation	252
4.21.2.1 CP	252
4.21.2.2 DEBUG_OPTIMIZE	252
4.21.2.3 RM	252
4.21.3 Function Documentation	252
4.21.3.1 optimize_best()	252
4.21.3.2 optimize_best_climbing()	253
4.21.3.3 optimize_climbing()	254
4.21.3.4 optimize_climbing_best()	255
4.21.3.5 optimize_climbing_sequential()	255
4.21.3.6 optimize_climbing_thread()	256
4.21.3.7 optimize_estimate_climbing_coordinates()	257
4.21.3.8 optimize_estimate_climbing_random()	257
4.21.3.9 optimize_free()	258
4.21.3.10 optimize_genetic()	258
4.21.3.11 optimize_genetic_objective()	259
4.21.3.12 optimize_input()	260
4.21.3.13 optimize_iterate()	261
4.21.3.14 optimize_merge()	262
4.21.3.15 optimize_merge_old()	263

4.21.3.16 optimize_MonteCarlo()	264
4.21.3.17 optimize_norm_euclidian()	265
4.21.3.18 optimize_norm_maximum()	265
4.21.3.19 optimize_norm_p()	266
4.21.3.20 optimize_norm_taxicab()	267
4.21.3.21 optimize_open()	268
4.21.3.22 optimize_orthogonal()	272
4.21.3.23 optimize_parse()	273
4.21.3.24 optimize_print()	275
4.21.3.25 optimize_refine()	275
4.21.3.26 optimize_save_old()	276
4.21.3.27 optimize_save_optimal()	277
4.21.3.28 optimize_save_variables()	278
4.21.3.29 optimize_sequential()	278
4.21.3.30 optimize_step()	279
4.21.3.31 optimize_step_climbing()	279
4.21.3.32 optimize_sweep()	280
4.21.3.33 optimize_synchronise()	281
4.21.3.34 optimize_thread()	282
4.21.4 Variable Documentation	283
4.21.4.1 nthreads_climbing	283
4.21.4.2 optimize	283
4.21.4.3 optimize_algorithm	284
4.21.4.4 optimize_estimate_climbing	284
4.21.4.5 optimize_norm	284
4.22 optimize.c	284
4.23 optimize.h File Reference	304
4.23.1 Detailed Description	304
4.23.2 Function Documentation	305
4.23.2.1 optimize_free()	305
4.23.2.2 optimize_open()	305
4.23.3 Variable Documentation	309
4.23.3.1 nthreads_climbing	309
4.23.3.2 optimize	309
4.24 optimize.h	310
4.25 tools.c File Reference	311
4.25.1 Detailed Description	312
4.25.2 Variable Documentation	312
4.25.2.1 error_message	312
4.25.2.2 main_window	312
4.25.2.3 show_pending	312
4.26 tools.c	313

4.27 tools.h File Reference . . . . .	313
4.27.1 Detailed Description . . . . .	314
4.27.2 Macro Definition Documentation . . . . .	314
4.27.2.1 ERROR_TYPE . . . . .	314
4.27.2.2 INFO_TYPE . . . . .	314
4.27.3 Variable Documentation . . . . .	315
4.27.3.1 error_message . . . . .	315
4.27.3.2 main_window . . . . .	315
4.27.3.3 show_pending . . . . .	315
4.28 tools.h . . . . .	315
4.29 variable.c File Reference . . . . .	316
4.29.1 Detailed Description . . . . .	317
4.29.2 Macro Definition Documentation . . . . .	317
4.29.2.1 DEBUG_VARIABLE . . . . .	317
4.29.3 Function Documentation . . . . .	317
4.29.3.1 variable_error() . . . . .	317
4.29.3.2 variable_free() . . . . .	318
4.29.3.3 variable_open_json() . . . . .	318
4.29.3.4 variable_open_xml() . . . . .	321
4.29.4 Variable Documentation . . . . .	324
4.29.4.1 format . . . . .	324
4.29.4.2 precision . . . . .	324
4.30 variable.c . . . . .	325
4.31 variable.h File Reference . . . . .	329
4.31.1 Detailed Description . . . . .	330
4.31.2 Enumeration Type Documentation . . . . .	330
4.31.2.1 Algorithm . . . . .	330
4.31.3 Function Documentation . . . . .	331
4.31.3.1 variable_error() . . . . .	331
4.31.3.2 variable_free() . . . . .	331
4.31.3.3 variable_open_json() . . . . .	332
4.31.3.4 variable_open_xml() . . . . .	334
4.31.4 Variable Documentation . . . . .	337
4.31.4.1 format . . . . .	337
4.31.4.2 precision . . . . .	337
4.32 variable.h . . . . .	338



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">Experiment</a>	Struct to define the experiment data . . . . .	<a href="#">5</a>
<a href="#">Input</a>	Struct to define the optimization input file . . . . .	<a href="#">6</a>
<a href="#">Optimize</a>	Struct to define the optimization ation data . . . . .	<a href="#">13</a>
<a href="#">Options</a>	Struct to define the options dialog . . . . .	<a href="#">25</a>
<a href="#">ParallelData</a>	Struct to pass to the GThreads parallelized function . . . . .	<a href="#">27</a>
<a href="#">Running</a>	Struct to define the running dialog . . . . .	<a href="#">28</a>
<a href="#">Variable</a>	Struct to define the variable data . . . . .	<a href="#">29</a>
<a href="#">Window</a>	Struct to define the main window . . . . .	<a href="#">32</a>



## Chapter 2

# File Index

### 2.1 File List

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

<a href="#">config.h</a>	Configuration header file . . . . .	57
<a href="#">experiment.c</a>	Source file to define the experiment data . . . . .	74
<a href="#">experiment.h</a>	Header file to define the experiment data . . . . .	87
<a href="#">input.c</a>	Source file to define the input functions . . . . .	96
<a href="#">input.h</a>	Header file to define the input functions . . . . .	125
<a href="#">interface.c</a>	Source file to define the graphical interface functions . . . . .	132
<a href="#">interface.h</a>	Header file to define the graphical interface functions . . . . .	221
<a href="#">main.c</a>	Main source file . . . . .	236
<a href="#">mpcotool.c</a>	Main function source file . . . . .	239
<a href="#">mpcotool.h</a>	Main function header file . . . . .	246
<a href="#">optimize.c</a>	Source file to define the optimization functions . . . . .	250
<a href="#">optimize.h</a>	Header file to define the optimization functions . . . . .	304
<a href="#">tools.c</a>	Source file to define some useful functions . . . . .	311
<a href="#">tools.h</a>	Header file to define some useful functions . . . . .	313
<a href="#">variable.c</a>	Source file to define the variable data . . . . .	316
<a href="#">variable.h</a>	Header file to define the variable data . . . . .	329



## Chapter 3

# Data Structure Documentation

### 3.1 Experiment Struct Reference

Struct to define the experiment data.

```
#include <experiment.h>
```

#### Data Fields

- char \* [name](#)  
*File name.*
- char \* [stencil](#) [[MAX\\_NINPUTS](#)]  
*Array of template names of input files.*
- double [weight](#)  
*Objective function weight.*
- unsigned int [ninputs](#)  
*Number of input files to the simulator.*
- unsigned int [template\\_flags](#)  
*Flags of template files.*

#### 3.1.1 Detailed Description

Struct to define the experiment data.

Definition at line [45](#) of file [experiment.h](#).

#### 3.1.2 Field Documentation

##### 3.1.2.1 name

```
char* Experiment::name
```

File name.

Definition at line [47](#) of file [experiment.h](#).

### 3.1.2.2 ninputs

```
unsigned int Experiment::ninputs
```

Number of input files to the simulator.

Definition at line 50 of file [experiment.h](#).

### 3.1.2.3 stencil

```
char* Experiment::stencil[MAX_NINPUTS]
```

Array of template names of input files.

Definition at line 48 of file [experiment.h](#).

### 3.1.2.4 template\_flags

```
unsigned int Experiment::template_flags
```

Flags of template files.

Definition at line 51 of file [experiment.h](#).

### 3.1.2.5 weight

```
double Experiment::weight
```

Objective function weight.

Definition at line 49 of file [experiment.h](#).

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

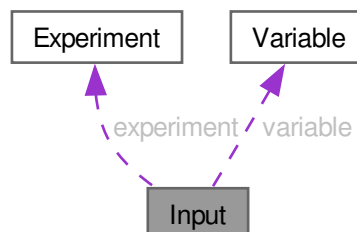
- [experiment.h](#)

## 3.2 Input Struct Reference

Struct to define the optimization input file.

```
#include <input.h>
```

Collaboration diagram for Input:



## Data Fields

- `Experiment * experiment`  
*Array or experiments.*
- `Variable * variable`  
*Array of variables.*
- `char * result`  
*Name of the result file.*
- `char * variables`  
*Name of the variables file.*
- `char * simulator`  
*Name of the simulator program.*
- `char * evaluator`  
*Name of the program to evaluate the objective function.*
- `char * cleaner`  
*Name of the cleaner program.*
- `char * directory`  
*Working directory.*
- `char * name`  
*Input data file name.*
- `double tolerance`  
*Algorithm tolerance.*
- `double mutation_ratio`  
*Mutation probability.*
- `double reproduction_ratio`  
*Reproduction probability.*
- `double adaptation_ratio`  
*Adaptation probability.*
- `double relaxation`  
*Relaxation parameter.*
- `double p`  
*Exponent of the P error norm.*
- `double threshold`  
*Threshold to finish the optimization.*
- `unsigned long int seed`  
*Seed of the pseudo-random numbers generator.*
- `unsigned int nvariables`  
*Variables number.*
- `unsigned int nexperiments`  
*Experiments number.*
- `unsigned int nsimulations`  
*Simulations number per experiment.*
- `unsigned int algorithm`  
*Algorithm type.*
- `unsigned int nsteps`  
*Number of steps to do the hill climbing method.*
- `unsigned int nfinal_steps`  
*Number of steps to do the hill climbing method at the final pass.*
- `unsigned int climbing`  
*Method to estimate the hill climbing.*
- `unsigned int nestimates`

- Number of simulations to estimate the hill climbing.*
  - unsigned int [niterations](#)  
*Number of algorithm iterations.*
  - unsigned int [nbest](#)  
*Number of best simulations.*
  - unsigned int [norm](#)  
*Error norm type.*
  - unsigned int [type](#)  
*Type of input file.*
  - unsigned int [template\\_flags](#)  
*Flags of template files.*

### 3.2.1 Detailed Description

Struct to define the optimization input file.

Definition at line 65 of file [input.h](#).

### 3.2.2 Field Documentation

#### 3.2.2.1 `adaptation_ratio`

```
double Input::adaptation_ratio
```

Adaptation probability.

Definition at line 80 of file [input.h](#).

#### 3.2.2.2 `algorithm`

```
unsigned int Input::algorithm
```

Algorithm type.

Definition at line 89 of file [input.h](#).

#### 3.2.2.3 `cleaner`

```
char* Input::cleaner
```

Name of the cleaner program.

Definition at line 74 of file [input.h](#).



#### 3.2.2.4 climbing

```
unsigned int Input::climbing
```

Method to estimate the hill climbing.

Definition at line 94 of file [input.h](#).

#### 3.2.2.5 directory

```
char* Input::directory
```

Working directory.

Definition at line 75 of file [input.h](#).

#### 3.2.2.6 evaluator

```
char* Input::evaluator
```

Name of the program to evaluate the objective function.

Definition at line 72 of file [input.h](#).

#### 3.2.2.7 experiment

```
Experiment* Input::experiment
```

Array or experiments.

Definition at line 67 of file [input.h](#).

#### 3.2.2.8 mutation\_ratio

```
double Input::mutation_ratio
```

Mutation probability.

Definition at line 78 of file [input.h](#).

#### 3.2.2.9 name

```
char* Input::name
```

[Input](#) data file name.

Definition at line 76 of file [input.h](#).

#### 3.2.2.10 nbest

```
unsigned int Input::nbest
```

Number of best simulations.

Definition at line 98 of file [input.h](#).

#### 3.2.2.11 nestimates

```
unsigned int Input::nestimates
```

Number of simulations to estimate the hill climbing.

Definition at line 95 of file [input.h](#).

#### 3.2.2.12 nexperiments

```
unsigned int Input::nexperiments
```

Experiments number.

Definition at line 87 of file [input.h](#).

#### 3.2.2.13 nfinal\_steps

```
unsigned int Input::nfinal_steps
```

Number of steps to do the hill climbing method at the final pass.

Definition at line 92 of file [input.h](#).

#### 3.2.2.14 niterations

```
unsigned int Input::niterations
```

Number of algorithm iterations.

Definition at line 97 of file [input.h](#).

#### 3.2.2.15 norm

```
unsigned int Input::norm
```

Error norm type.

Definition at line 99 of file [input.h](#).

### 3.2.2.16 nsimulations

```
unsigned int Input::nsimulations
```

Simulations number per experiment.

Definition at line 88 of file [input.h](#).

### 3.2.2.17 nsteps

```
unsigned int Input::nsteps
```

Number of steps to do the hill climbing method.

Definition at line 90 of file [input.h](#).

### 3.2.2.18 nvariables

```
unsigned int Input::nvariables
```

Variables number.

Definition at line 86 of file [input.h](#).

### 3.2.2.19 p

```
double Input::p
```

Exponent of the P error norm.

Definition at line 82 of file [input.h](#).

### 3.2.2.20 relaxation

```
double Input::relaxation
```

Relaxation parameter.

Definition at line 81 of file [input.h](#).

### 3.2.2.21 reproduction\_ratio

```
double Input::reproduction_ratio
```

Reproduction probability.

Definition at line 79 of file [input.h](#).

#### 3.2.2.22 result

```
char* Input::result
```

Name of the result file.

Definition at line 69 of file [input.h](#).

#### 3.2.2.23 seed

```
unsigned long int Input::seed
```

Seed of the pseudo-random numbers generator.

Definition at line 84 of file [input.h](#).

#### 3.2.2.24 simulator

```
char* Input::simulator
```

Name of the simulator program.

Definition at line 71 of file [input.h](#).

#### 3.2.2.25 template\_flags

```
unsigned int Input::template_flags
```

Flags of template files.

Definition at line 101 of file [input.h](#).

#### 3.2.2.26 threshold

```
double Input::threshold
```

Threshold to finish the optimization.

Definition at line 83 of file [input.h](#).

#### 3.2.2.27 tolerance

```
double Input::tolerance
```

Algorithm tolerance.

Definition at line 77 of file [input.h](#).

#### 3.2.2.28 type

```
unsigned int Input::type
```

Type of input file.

Definition at line 100 of file [input.h](#).

#### 3.2.2.29 variable

```
Variable* Input::variable
```

Array of variables.

Definition at line 68 of file [input.h](#).

#### 3.2.2.30 variables

```
char* Input::variables
```

Name of the variables file.

Definition at line 70 of file [input.h](#).

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

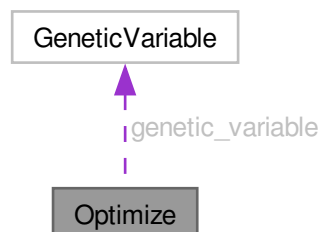
- [input.h](#)

## 3.3 Optimize Struct Reference

Struct to define the optimization ation data.

```
#include <optimize.h>
```

Collaboration diagram for Optimize:



## Data Fields

- `GMappedFile ** file [MAX_NINPUTS]`  
*Matrix of input template files.*
- `char ** experiment`  
*Array of experimental data file names.*
- `char ** label`  
*Array of variable names.*
- `gsl_rng * rng`  
*GSL random number generator.*
- **GeneticVariable** \* `genetic_variable`  
*Array of variables for the genetic algorithm.*
- `FILE * file_result`  
*Result file.*
- `FILE * file_variables`  
*Variables file.*
- `char * result`  
*Name of the result file.*
- `char * variables`  
*Name of the variables file.*
- `char * simulator`  
*Name of the simulator program.*
- `char * evaluator`  
*Name of the program to evaluate the objective function.*
- `char * cleaner`  
*Name of the cleaner program.*
- `double * value`  
*Array of variable values.*
- `double * rangemin`  
*Array of minimum variable values.*
- `double * rangemax`  
*Array of maximum variable values.*
- `double * rangeminabs`  
*Array of absolute minimum variable values.*
- `double * rangemaxabs`  
*Array of absolute maximum variable values.*
- `double * error_best`  
*Array of the best minimum errors.*
- `double * weight`  
*Array of the experiment weights.*
- `double * step`  
*Array of hill climbing method step sizes.*
- `double * climbing`  
*Vector of hill climbing estimation.*
- `double * value_old`  
*Array of the best variable values on the previous step.*
- `double * error_old`  
*Array of the best minimum errors on the previous step.*
- `unsigned int * precision`  
*Array of variable precisions.*
- `unsigned int * nsweeps`

- Array of sweeps of the sweep algorithm.*
- `unsigned int * nbits`
- Array of bits number of the genetic algorithm.*
- `unsigned int * thread`
- Array of simulation numbers to calculate on the thread.*
- `unsigned int * thread_climbing`
- `unsigned int * simulation_best`
- Array of best simulation numbers.*
- `double tolerance`
- Algorithm tolerance.*
- `double mutation_ratio`
- Mutation probability.*
- `double reproduction_ratio`
- Reproduction probability.*
- `double adaptation_ratio`
- Adaptation probability.*
- `double relaxation`
- Relaxation parameter.*
- `double calculation_time`
- Calculation time.*
- `double p`
- Exponent of the P error norm.*
- `double threshold`
- Threshold to finish the optimization.*
- `unsigned long int seed`
- Seed of the pseudo-random numbers generator.*
- `unsigned int nvariables`
- Variables number.*
- `unsigned int nexperiments`
- Experiments number.*
- `unsigned int ninputs`
- Number of input files to the simulator.*
- `unsigned int nsimulations`
- Simulations number per experiment.*
- `unsigned int nsteps`
- Number of steps for the hill climbing method.*
- `unsigned int nfinal_steps`
- Number of steps to do the hill climbing method at the final pass.*
- `unsigned int nestimates`
- Number of simulations to estimate the climbing.*
- `unsigned int algorithm`
- Algorithm type.*
- `unsigned int nstart`
- Beginning simulation number of the task.*
- `unsigned int nend`
- Ending simulation number of the task.*
- `unsigned int nstart_climbing`
- Beginning simulation number of the task for the hill climbing method.*
- `unsigned int nend_climbing`
- Ending simulation number of the task for the hill climbing method.*
- `unsigned int niterations`

- Number of algorithm iterations.*
  - `unsigned int nbest`  
*Number of best simulations.*
  - `unsigned int nsaveds`  
*Number of saved simulations.*
  - `unsigned int stop`  
*To stop the simulations.*
  - `unsigned int template_flags`  
*Flags of template files.*
  - `int mpi_rank`  
*Number of MPI task.*

### 3.3.1 Detailed Description

Struct to define the optimization ation data.

Definition at line 45 of file [optimize.h](#).

### 3.3.2 Field Documentation

#### 3.3.2.1 adaptation\_ratio

`double` `Optimize::adaptation_ratio`

Adaptation probability.

Definition at line 87 of file [optimize.h](#).

#### 3.3.2.2 algorithm

`unsigned int` `Optimize::algorithm`

Algorithm type.

Definition at line 104 of file [optimize.h](#).

#### 3.3.2.3 calculation\_time

`double` `Optimize::calculation_time`

Calculation time.

Definition at line 89 of file [optimize.h](#).



#### 3.3.2.4 cleaner

```
char* Optimize::cleaner
```

Name of the cleaner program.

Definition at line 60 of file [optimize.h](#).

#### 3.3.2.5 climbing

```
double* Optimize::climbing
```

Vector of hill climbing estimation.

Definition at line 69 of file [optimize.h](#).

#### 3.3.2.6 error\_best

```
double* Optimize::error_best
```

Array of the best minimum errors.

Definition at line 66 of file [optimize.h](#).

#### 3.3.2.7 error\_old

```
double* Optimize::error_old
```

Array of the best minimum errors on the previous step.

Definition at line 72 of file [optimize.h](#).

#### 3.3.2.8 evaluator

```
char* Optimize::evaluator
```

Name of the program to evaluate the objective function.

Definition at line 58 of file [optimize.h](#).

#### 3.3.2.9 experiment

```
char** Optimize::experiment
```

Array of experimental data file names.

Definition at line 48 of file [optimize.h](#).

### 3.3.2.10 file

```
GMappedFile** Optimize::file[MAX_NINPUTS]
```

Matrix of input template files.

Definition at line 47 of file [optimize.h](#).

### 3.3.2.11 file\_result

```
FILE* Optimize::file_result
```

Result file.

Definition at line 53 of file [optimize.h](#).

### 3.3.2.12 file\_variables

```
FILE* Optimize::file_variables
```

Variables file.

Definition at line 54 of file [optimize.h](#).

### 3.3.2.13 genetic\_variable

```
GeneticVariable* Optimize::genetic_variable
```

Array of variables for the genetic algorithm.

Definition at line 51 of file [optimize.h](#).

### 3.3.2.14 label

```
char** Optimize::label
```

Array of variable names.

Definition at line 49 of file [optimize.h](#).

### 3.3.2.15 mpi\_rank

```
int Optimize::mpi_rank
```

Number of MPI task.

Definition at line 117 of file [optimize.h](#).

### 3.3.2.16 mutation\_ratio

`double` `Optimize::mutation_ratio`

Mutation probability.

Definition at line 85 of file [optimize.h](#).

### 3.3.2.17 nbest

`unsigned int` `Optimize::nbest`

Number of best simulations.

Definition at line 112 of file [optimize.h](#).

### 3.3.2.18 nbits

`unsigned int*` `Optimize::nbits`

Array of bits number of the genetic algorithm.

Definition at line 76 of file [optimize.h](#).

### 3.3.2.19 nend

`unsigned int` `Optimize::nend`

Ending simulation number of the task.

Definition at line 106 of file [optimize.h](#).

### 3.3.2.20 nend\_climbing

`unsigned int` `Optimize::nend_climbing`

Ending simulation number of the task for the hill climbing method.

Definition at line 109 of file [optimize.h](#).

### 3.3.2.21 nestimates

`unsigned int` `Optimize::nestimates`

Number of simulations to estimate the climbing.

Definition at line 102 of file [optimize.h](#).

#### 3.3.2.22 nexperiments

`unsigned int` Optimize::nexperiments

Experiments number.

Definition at line 95 of file [optimize.h](#).

#### 3.3.2.23 nfinal\_steps

`unsigned int` Optimize::nfinal\_steps

Number of steps to do the hill climbing method at the final pass.

Definition at line 100 of file [optimize.h](#).

#### 3.3.2.24 ninputs

`unsigned int` Optimize::ninputs

Number of input files to the simulator.

Definition at line 96 of file [optimize.h](#).

#### 3.3.2.25 niterations

`unsigned int` Optimize::niterations

Number of algorithm iterations.

Definition at line 111 of file [optimize.h](#).

#### 3.3.2.26 nsaveds

`unsigned int` Optimize::nsaveds

Number of saved simulations.

Definition at line 113 of file [optimize.h](#).

#### 3.3.2.27 nsimulations

`unsigned int` Optimize::nsimulations

Simulations number per experiment.

Definition at line 97 of file [optimize.h](#).

#### 3.3.2.28 nstart

```
unsigned int Optimize::nstart
```

Beginning simulation number of the task.

Definition at line 105 of file [optimize.h](#).

#### 3.3.2.29 nstart\_climbing

```
unsigned int Optimize::nstart_climbing
```

Beginning simulation number of the task for the hill climbing method.

Definition at line 107 of file [optimize.h](#).

#### 3.3.2.30 nsteps

```
unsigned int Optimize::nsteps
```

Number of steps for the hill climbing method.

Definition at line 98 of file [optimize.h](#).

#### 3.3.2.31 nsweeps

```
unsigned int* Optimize::nsweeps
```

Array of sweeps of the sweep algorithm.

Definition at line 75 of file [optimize.h](#).

#### 3.3.2.32 nvariables

```
unsigned int Optimize::nvariables
```

Variables number.

Definition at line 94 of file [optimize.h](#).

#### 3.3.2.33 p

```
double Optimize::p
```

Exponent of the P error norm.

Definition at line 90 of file [optimize.h](#).

#### 3.3.2.34 precision

```
unsigned int* Optimize::precision
```

Array of variable precisions.

Definition at line 74 of file [optimize.h](#).

#### 3.3.2.35 rangemax

```
double* Optimize::rangemax
```

Array of maximum variable values.

Definition at line 63 of file [optimize.h](#).

#### 3.3.2.36 rangemaxabs

```
double* Optimize::rangemaxabs
```

Array of absolute maximum variable values.

Definition at line 65 of file [optimize.h](#).

#### 3.3.2.37 rangemin

```
double* Optimize::rangemin
```

Array of minimum variable values.

Definition at line 62 of file [optimize.h](#).

#### 3.3.2.38 rangeminabs

```
double* Optimize::rangeminabs
```

Array of absolute minimum variable values.

Definition at line 64 of file [optimize.h](#).

#### 3.3.2.39 relaxation

```
double Optimize::relaxation
```

Relaxation parameter.

Definition at line 88 of file [optimize.h](#).

#### 3.3.2.40 reproduction\_ratio

`double` Optimize::reproduction\_ratio

Reproduction probability.

Definition at line 86 of file [optimize.h](#).

#### 3.3.2.41 result

`char*` Optimize::result

Name of the result file.

Definition at line 55 of file [optimize.h](#).

#### 3.3.2.42 rng

`gsl_rng*` Optimize::rng

GSL random number generator.

Definition at line 50 of file [optimize.h](#).

#### 3.3.2.43 seed

`unsigned long int` Optimize::seed

Seed of the pseudo-random numbers generator.

Definition at line 92 of file [optimize.h](#).

#### 3.3.2.44 simulation\_best

`unsigned int*` Optimize::simulation\_best

Array of best simulation numbers.

Definition at line 83 of file [optimize.h](#).

#### 3.3.2.45 simulator

`char*` Optimize::simulator

Name of the simulator program.

Definition at line 57 of file [optimize.h](#).

#### 3.3.2.46 step

`double*` `Optimize::step`

Array of hill climbing method step sizes.

Definition at line 68 of file [optimize.h](#).

#### 3.3.2.47 stop

`unsigned int` `Optimize::stop`

To stop the simulations.

Definition at line 114 of file [optimize.h](#).

#### 3.3.2.48 template\_flags

`unsigned int` `Optimize::template_flags`

Flags of template files.

Definition at line 115 of file [optimize.h](#).

#### 3.3.2.49 thread

`unsigned int*` `Optimize::thread`

Array of simulation numbers to calculate on the thread.

Definition at line 78 of file [optimize.h](#).

#### 3.3.2.50 thread\_climbing

`unsigned int*` `Optimize::thread_climbing`

Array of simulation numbers to calculate on the thread for the hill climbing method.

Definition at line 80 of file [optimize.h](#).

#### 3.3.2.51 threshold

`double` `Optimize::threshold`

Threshold to finish the optimization.

Definition at line 91 of file [optimize.h](#).



### 3.3.2.52 tolerance

```
double Optimize::tolerance
```

Algorithm tolerance.

Definition at line 84 of file [optimize.h](#).

### 3.3.2.53 value

```
double* Optimize::value
```

Array of variable values.

Definition at line 61 of file [optimize.h](#).

### 3.3.2.54 value\_old

```
double* Optimize::value_old
```

Array of the best variable values on the previous step.

Definition at line 70 of file [optimize.h](#).

### 3.3.2.55 variables

```
char* Optimize::variables
```

Name of the variables file.

Definition at line 56 of file [optimize.h](#).

### 3.3.2.56 weight

```
double* Optimize::weight
```

Array of the experiment weights.

Definition at line 67 of file [optimize.h](#).

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

- [optimize.h](#)

## 3.4 Options Struct Reference

Struct to define the options dialog.

```
#include <interface.h>
```

## Data Fields

- GtkDialog \* [dialog](#)  
*Main GtkDialog.*
- GtkGrid \* [grid](#)  
*Main GtkGrid.*
- GtkLabel \* [label\\_seed](#)  
*Pseudo-random numbers generator seed GtkLabel.*
- GtkSpinButton \* [spin\\_seed](#)  
*Pseudo-random numbers generator seed GtkSpinButton.*
- GtkLabel \* [label\\_threads](#)  
*Threads number GtkLabel.*
- GtkSpinButton \* [spin\\_threads](#)  
*Threads number GtkSpinButton.*
- GtkLabel \* [label\\_climbing](#)  
*Climbing threads number GtkLabel.*
- GtkSpinButton \* [spin\\_climbing](#)  
*Climbing threads number GtkSpinButton.*

### 3.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 48 of file [interface.h](#).

### 3.4.2 Field Documentation

#### 3.4.2.1 dialog

```
GtkDialog* Options::dialog
```

Main GtkDialog.

Definition at line 50 of file [interface.h](#).

#### 3.4.2.2 grid

```
GtkGrid* Options::grid
```

Main GtkGrid.

Definition at line 51 of file [interface.h](#).

#### 3.4.2.3 label\_climbing

```
GtkLabel* Options::label_climbing
```

Climbing threads number GtkLabel.

Definition at line 58 of file [interface.h](#).

#### 3.4.2.4 label\_seed

```
GtkLabel* Options::label_seed
```

Pseudo-random numbers generator seed GtkLabel.

Definition at line 52 of file [interface.h](#).

#### 3.4.2.5 label\_threads

```
GtkLabel* Options::label_threads
```

Threads number GtkLabel.

Definition at line 56 of file [interface.h](#).

#### 3.4.2.6 spin\_climbing

```
GtkSpinButton* Options::spin_climbing
```

Climbing threads number GtkSpinButton.

Definition at line 59 of file [interface.h](#).

#### 3.4.2.7 spin\_seed

```
GtkSpinButton* Options::spin_seed
```

Pseudo-random numbers generator seed GtkSpinButton.

Definition at line 54 of file [interface.h](#).

#### 3.4.2.8 spin\_threads

```
GtkSpinButton* Options::spin_threads
```

Threads number GtkSpinButton.

Definition at line 57 of file [interface.h](#).

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

- [interface.h](#)

## 3.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

```
#include <optimize.h>
```

## Data Fields

- unsigned int [thread](#)  
*Thread number.*

### 3.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line [125](#) of file [optimize.h](#).

### 3.5.2 Field Documentation

#### 3.5.2.1 thread

```
unsigned int ParallelData::thread
```

Thread number.

Definition at line [127](#) of file [optimize.h](#).

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

- [optimize.h](#)

## 3.6 Running Struct Reference

Struct to define the running dialog.

```
#include <interface.h>
```

## Data Fields

- GtkWidget \* [dialog](#)  
*Main GtkDialog.*
- GtkWidget \* [label](#)  
*Label GtkWidget.*
- GtkWidget \* [spinner](#)  
*Animation GtkWidget.*
- GtkWidget \* [grid](#)  
*Grid GtkWidget.*

### 3.6.1 Detailed Description

Struct to define the running dialog.

Definition at line [66](#) of file [interface.h](#).

## 3.6.2 Field Documentation

### 3.6.2.1 dialog

```
GtkDialog* Running::dialog
```

Main GtkDialog.

Definition at line 68 of file [interface.h](#).

### 3.6.2.2 grid

```
GtkGrid* Running::grid
```

Grid GtkGrid.

Definition at line 71 of file [interface.h](#).

### 3.6.2.3 label

```
GtkLabel* Running::label
```

Label GtkLabel.

Definition at line 69 of file [interface.h](#).

### 3.6.2.4 spinner

```
GtkSpinner* Running::spinner
```

Animation GtkSpinner.

Definition at line 70 of file [interface.h](#).

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

- [interface.h](#)

## 3.7 Variable Struct Reference

Struct to define the variable data.

```
#include <variable.h>
```

## Data Fields

- char \* [name](#)  
*Variable name.*
- double [rangemin](#)  
*Minimum variable value.*
- double [rangemax](#)  
*Maximum variable value.*
- double [rangeminabs](#)  
*Absolute minimum variable value.*
- double [rangemaxabs](#)  
*Absolute maximum variable value.*
- double [step](#)  
*Hill climbing method step size.*
- unsigned int [precision](#)  
*Variable precision.*
- unsigned int [nsweeps](#)  
*Sweeps of the sweep algorithm.*
- unsigned int [nbits](#)  
*Bits number of the genetic algorithm.*

### 3.7.1 Detailed Description

Struct to define the variable data.

Definition at line 54 of file [variable.h](#).

### 3.7.2 Field Documentation

#### 3.7.2.1 name

```
char* Variable::name
```

[Variable](#) name.

Definition at line 56 of file [variable.h](#).

#### 3.7.2.2 nbits

```
unsigned int Variable::nbits
```

Bits number of the genetic algorithm.

Definition at line 64 of file [variable.h](#).

### 3.7.2.3 nsweeps

```
unsigned int Variable::nsweeps
```

Sweeps of the sweep algorithm.

Definition at line 63 of file [variable.h](#).

### 3.7.2.4 precision

```
unsigned int Variable::precision
```

[Variable](#) precision.

Definition at line 62 of file [variable.h](#).

### 3.7.2.5 rangemax

```
double Variable::rangemax
```

Maximum variable value.

Definition at line 58 of file [variable.h](#).

### 3.7.2.6 rangemaxabs

```
double Variable::rangemaxabs
```

Absolute maximum variable value.

Definition at line 60 of file [variable.h](#).

### 3.7.2.7 rangemin

```
double Variable::rangemin
```

Minimum variable value.

Definition at line 57 of file [variable.h](#).

### 3.7.2.8 rangeminabs

```
double Variable::rangeminabs
```

Absolute minimum variable value.

Definition at line 59 of file [variable.h](#).

### 3.7.2.9 step

```
double Variable::step
```

Hill climbing method step size.

Definition at line 61 of file [variable.h](#).

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

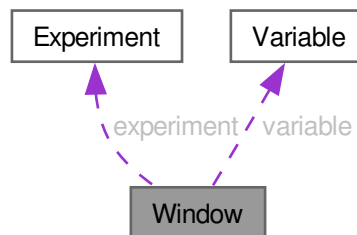
- [variable.h](#)

## 3.8 Window Struct Reference

Struct to define the main window.

```
#include <interface.h>
```

Collaboration diagram for Window:



### Data Fields

- GtkWidget \* [window](#)  
*Main GtkWidget.*
- GtkWidget \* [grid](#)  
*Main GtkWidget.*
- GtkWidget \* [box\\_buttons](#)  
*GtkWidget to store the main buttons.*
- GtkWidget \* [button\\_open](#)  
*Open GtkWidget.*
- GtkWidget \* [button\\_save](#)  
*Save GtkWidget.*
- GtkWidget \* [button\\_run](#)  
*Run GtkWidget.*
- GtkWidget \* [button\\_options](#)  
*Options GtkWidget.*



- `GtkButton * button_help`  
*Help GtkButton.*
- `GtkButton * button_about`  
*Help GtkButton.*
- `GtkButton * button_exit`  
*Exit GtkButton.*
- `GtkGrid * grid_files`  
*Files GtkGrid.*
- `GtkLabel * label_simulator`  
*Simulator program GtkLabel.*
- `GtkButton * button_simulator`  
*Simulator program GtkButton.*
- `GtkCheckButton * check_evaluator`  
*Evaluator program GtkCheckButton.*
- `GtkButton * button_evaluator`  
*Evaluator program GtkButton.*
- `GtkCheckButton * check_cleaner`  
*Cleaner program GtkCheckButton.*
- `GtkButton * button_cleaner`  
*Cleaner program GtkButton.*
- `GtkLabel * label_result`  
*Result file GtkLabel.*
- `GtkEntry * entry_result`  
*Result file GtkEntry.*
- `GtkLabel * label_variables`  
*Variables file GtkLabel.*
- `GtkEntry * entry_variables`  
*Variables file GtkEntry.*
- `GtkFrame * frame_norm`  
*GtkFrame to set the error norm.*
- `GtkGrid * grid_norm`  
*GtkGrid to set the error norm.*
- `GtkRadioButton * button_norm [NNORMS]`  
*Array of GtkRadioButtons to set the error norm.*
- `GtkLabel * label_p`  
*GtkLabel to set the p parameter.*
- `GtkSpinButton * spin_p`  
*GtkSpinButton to set the p parameter.*
- `GtkScrolledWindow * scrolled_p`  
*GtkScrolledWindow to set the p parameter.*
- `GtkFrame * frame_algorithm`  
*GtkFrame to set the algorithm.*
- `GtkGrid * grid_algorithm`  
*GtkGrid to set the algorithm.*
- `GtkRadioButton * button_algorithm [NALGORITHMS]`  
*Array of GtkRadioButtons to set the algorithm.*
- `GtkLabel * label_simulations`  
*GtkLabel to set the simulations number.*
- `GtkSpinButton * spin_simulations`  
*GtkSpinButton to set the simulations number.*
- `GtkLabel * label_iterations`

- GtkLabel to set the iterations number.*
- GtkSpinButton \* [spin\\_iterations](#)  
*GtkSpinButton to set the iterations number.*
- GtkLabel \* [label\\_tolerance](#)  
*GtkLabel to set the tolerance.*
- GtkSpinButton \* [spin\\_tolerance](#)  
*GtkSpinButton to set the tolerance.*
- GtkLabel \* [label\\_bests](#)  
*GtkLabel to set the best number.*
- GtkSpinButton \* [spin\\_bests](#)  
*GtkSpinButton to set the best number.*
- GtkLabel \* [label\\_population](#)  
*GtkLabel to set the population number.*
- GtkSpinButton \* [spin\\_population](#)  
*GtkSpinButton to set the population number.*
- GtkLabel \* [label\\_generations](#)  
*GtkLabel to set the generations number.*
- GtkSpinButton \* [spin\\_generations](#)  
*GtkSpinButton to set the generations number.*
- GtkLabel \* [label\\_mutation](#)  
*GtkLabel to set the mutation ratio.*
- GtkSpinButton \* [spin\\_mutation](#)  
*GtkSpinButton to set the mutation ratio.*
- GtkLabel \* [label\\_reproduction](#)  
*GtkLabel to set the reproduction ratio.*
- GtkSpinButton \* [spin\\_reproduction](#)  
*GtkSpinButton to set the reproduction ratio.*
- GtkLabel \* [label\\_adaptation](#)  
*GtkLabel to set the adaptation ratio.*
- GtkSpinButton \* [spin\\_adaptation](#)  
*GtkSpinButton to set the adaptation ratio.*
- GtkCheckBox \* [check\\_climbing](#)  
*GtkCheckBox to check running the hill climbing method.*
- GtkGrid \* [grid\\_climbing](#)  
*GtkGrid to pack the hill climbing method widgets.*
- GtkRadioButton \* [button\\_climbing](#) [NCLIMBINGS]  
*Array of GtkRadioButtons array to set the hill climbing method.*
- GtkLabel \* [label\\_steps](#)  
*GtkLabel to set the steps number.*
- GtkSpinButton \* [spin\\_steps](#)  
*GtkSpinButton to set the steps number.*
- GtkLabel \* [label\\_final\\_steps](#)  
*GtkLabel to set the final steps number.*
- GtkSpinButton \* [spin\\_final\\_steps](#)  
*GtkSpinButton to set the final steps number.*
- GtkLabel \* [label\\_estimates](#)  
*GtkLabel to set the estimates number.*
- GtkSpinButton \* [spin\\_estimates](#)  
*GtkSpinButton to set the estimates number.*
- GtkLabel \* [label\\_relaxation](#)  
*GtkLabel to set the relaxation parameter.*

- GtkSpinButton \* [spin\\_relaxation](#)  
*GtkSpinButton to set the relaxation parameter.*
- GtkLabel \* [label\\_threshold](#)  
*GtkLabel to set the threshold.*
- GtkSpinButton \* [spin\\_threshold](#)  
*GtkSpinButton to set the threshold.*
- GtkScrolledWindow \* [scrolled\\_threshold](#)  
*GtkScrolledWindow to set the threshold.*
- GtkFrame \* [frame\\_variable](#)  
*Variable GtkFrame.*
- GtkGrid \* [grid\\_variable](#)  
*Variable GtkGrid.*
- GtkComboBoxText \* [combo\\_variable](#)  
*GtkComboBoxEntry to select a variable.*
- GtkButton \* [button\\_add\\_variable](#)  
*GtkButton to add a variable.*
- GtkButton \* [button\\_remove\\_variable](#)  
*GtkButton to remove a variable.*
- GtkLabel \* [label\\_variable](#)  
*Variable GtkLabel.*
- GtkEntry \* [entry\\_variable](#)  
*GtkEntry to set the variable name.*
- GtkLabel \* [label\\_min](#)  
*Minimum GtkLabel.*
- GtkSpinButton \* [spin\\_min](#)  
*Minimum GtkSpinButton.*
- GtkScrolledWindow \* [scrolled\\_min](#)  
*Minimum GtkScrolledWindow.*
- GtkLabel \* [label\\_max](#)  
*Maximum GtkLabel.*
- GtkSpinButton \* [spin\\_max](#)  
*Maximum GtkSpinButton.*
- GtkScrolledWindow \* [scrolled\\_max](#)  
*Maximum GtkScrolledWindow.*
- GtkCheckButton \* [check\\_minabs](#)  
*Absolute minimum GtkCheckButton.*
- GtkSpinButton \* [spin\\_minabs](#)  
*Absolute minimum GtkSpinButton.*
- GtkScrolledWindow \* [scrolled\\_minabs](#)  
*Absolute minimum GtkScrolledWindow.*
- GtkCheckButton \* [check\\_maxabs](#)  
*Absolute maximum GtkCheckButton.*
- GtkSpinButton \* [spin\\_maxabs](#)  
*Absolute maximum GtkSpinButton.*
- GtkScrolledWindow \* [scrolled\\_maxabs](#)  
*Absolute maximum GtkScrolledWindow.*
- GtkLabel \* [label\\_precision](#)  
*Precision GtkLabel.*
- GtkSpinButton \* [spin\\_precision](#)  
*Precision digits GtkSpinButton.*
- GtkLabel \* [label\\_sweeps](#)

- Sweeps number GtkLabel.*
- GtkWidget \* [spin\\_sweeps](#)
  - Sweeps number GtkSpinButton.*
- GtkWidget \* [label\\_bits](#)
  - Bits number GtkLabel.*
- GtkWidget \* [spin\\_bits](#)
  - Bits number GtkSpinButton.*
- GtkWidget \* [label\\_step](#)
  - GtkLabel to set the step.*
- GtkWidget \* [spin\\_step](#)
  - GtkSpinButton to set the step.*
- GtkWidget \* [scrolled\\_step](#)
  - step GtkWidget.*
- GtkWidget \* [frame\\_experiment](#)
  - Experiment GtkWidget.*
- GtkWidget \* [grid\\_experiment](#)
  - Experiment GtkWidget.*
- GtkWidget \* [combo\\_experiment](#)
  - Experiment GtkWidgetEntry.*
- GtkWidget \* [button\\_add\\_experiment](#)
  - GtkButton to add a experiment.*
- GtkWidget \* [button\\_remove\\_experiment](#)
  - GtkButton to remove a experiment.*
- GtkWidget \* [label\\_experiment](#)
  - Experiment GtkWidget.*
- GtkWidget \* [button\\_experiment](#)
  - GtkButton to set the experimental data file.*
- GtkWidget \* [label\\_weight](#)
  - Weight GtkWidget.*
- GtkWidget \* [spin\\_weight](#)
  - Weight GtkSpinButton.*
- GtkWidget \* [check\\_template](#) [MAX\_NINPUTS]
  - Array of GtkCheckButtons to set the input templates.*
- GtkWidget \* [button\\_template](#) [MAX\_NINPUTS]
  - Array of GtkButtons to set the input templates.*
- GdkPixbuf \* [logo](#)
  - Logo GdkPixbuf.*
- [Experiment](#) \* [experiment](#)
  - Array of experiments data.*
- [Variable](#) \* [variable](#)
  - Array of variables data.*
- char \* [application\\_directory](#)
  - Application directory.*
- gulong [id\\_experiment](#)
  - Identifier of the combo\_experiment signal.*
- gulong [id\\_experiment\\_name](#)
  - Identifier of the button\_experiment signal.*
- gulong [id\\_variable](#)
  - Identifier of the combo\_variable signal.*
- gulong [id\\_variable\\_label](#)
  - Identifier of the entry\_variable signal.*

- gulong [id\\_template](#) [[MAX\\_NINPUTS](#)]  
*Array of identifiers of the check\_template signal.*
- gulong [id\\_input](#) [[MAX\\_NINPUTS](#)]  
*Array of identifiers of the button\_template signal.*
- unsigned int [nexperiments](#)  
*Number of experiments.*
- unsigned int [nvariables](#)  
*Number of variables.*

### 3.8.1 Detailed Description

Struct to define the main window.

Definition at line [78](#) of file [interface.h](#).

### 3.8.2 Field Documentation

#### 3.8.2.1 application\_directory

```
char* Window::application_directory
```

Application directory.

Definition at line [220](#) of file [interface.h](#).

#### 3.8.2.2 box\_buttons

```
GtkBox* Window::box_buttons
```

GtkBox to store the main buttons.

Definition at line [82](#) of file [interface.h](#).

#### 3.8.2.3 button\_about

```
GtkButton* Window::button_about
```

Help GtkButton.

Definition at line [88](#) of file [interface.h](#).

#### 3.8.2.4 button\_add\_experiment

```
GtkButton* Window::button_add_experiment
```

GtkButton to add a experiment.

Definition at line [206](#) of file [interface.h](#).

#### 3.8.2.5 button\_add\_variable

```
GtkButton* Window::button_add_variable
```

GtkButton to add a variable.

Definition at line 178 of file [interface.h](#).

#### 3.8.2.6 button\_algorithm

```
GtkRadioButton* Window::button_algorithm[NALGORITHMS]
```

Array of GtkRadioButtons to set the algorithm.

Definition at line 117 of file [interface.h](#).

#### 3.8.2.7 button\_cleaner

```
GtkButton* Window::button_cleaner
```

Cleaner program GtkButton.

Definition at line 96 of file [interface.h](#).

#### 3.8.2.8 button\_climbing

```
GtkRadioButton* Window::button_climbing[NCLIMBINGS]
```

Array of GtkRadioButtons array to set the hill climbing method.

Definition at line 152 of file [interface.h](#).

#### 3.8.2.9 button\_evaluator

```
GtkButton* Window::button_evaluator
```

Evaluator program GtkButton.

Definition at line 94 of file [interface.h](#).

#### 3.8.2.10 button\_exit

```
GtkButton* Window::button_exit
```

Exit GtkButton.

Definition at line 89 of file [interface.h](#).

### 3.8.2.11 button\_experiment

GtkButton\* Window::button\_experiment

GtkButton to set the experimental data file.

Definition at line 209 of file [interface.h](#).

### 3.8.2.12 button\_help

GtkButton\* Window::button\_help

Help GtkButton.

Definition at line 87 of file [interface.h](#).

### 3.8.2.13 button\_norm

GtkRadioButton\* Window::button\_norm[NNORMS]

Array of GtkRadioButtons to set the error norm.

Definition at line 104 of file [interface.h](#).

### 3.8.2.14 button\_open

GtkButton\* Window::button\_open

Open GtkButton.

Definition at line 83 of file [interface.h](#).

### 3.8.2.15 button\_options

GtkButton\* Window::button\_options

[Options](#) GtkButton.

Definition at line 86 of file [interface.h](#).

### 3.8.2.16 button\_remove\_experiment

GtkButton\* Window::button\_remove\_experiment

GtkButton to remove a experiment.

Definition at line 207 of file [interface.h](#).

### 3.8.2.17 button\_remove\_variable

GtkButton\* Window::button\_remove\_variable

GtkButton to remove a variable.

Definition at line 179 of file [interface.h](#).

### 3.8.2.18 button\_run

GtkButton\* Window::button\_run

Run GtkButton.

Definition at line 85 of file [interface.h](#).

### 3.8.2.19 button\_save

GtkButton\* Window::button\_save

Save GtkButton.

Definition at line 84 of file [interface.h](#).

### 3.8.2.20 button\_simulator

GtkButton\* Window::button\_simulator

Simulator program GtkButton.

Definition at line 92 of file [interface.h](#).

### 3.8.2.21 button\_template

GtkButton\* Window::button\_template[[MAX\\_NINPUTS](#)]

Array of GtkButtons to set the input templates.

Definition at line 215 of file [interface.h](#).

### 3.8.2.22 check\_cleaner

GtkCheckButton\* Window::check\_cleaner

Cleaner program GtkCheckButton.

Definition at line 95 of file [interface.h](#).



### 3.8.2.23 check\_climbing

```
GtkCheckButton* Window::check_climbing
```

GtkCheckButton to check running the hill climbing method.

Definition at line 147 of file [interface.h](#).

### 3.8.2.24 check\_evaluator

```
GtkCheckButton* Window::check_evaluator
```

Evaluator program GtkCheckButton.

Definition at line 93 of file [interface.h](#).

### 3.8.2.25 check\_maxabs

```
GtkCheckButton* Window::check_maxabs
```

Absolute maximum GtkCheckButton.

Definition at line 191 of file [interface.h](#).

### 3.8.2.26 check\_minabs

```
GtkCheckButton* Window::check_minabs
```

Absolute minimum GtkCheckButton.

Definition at line 188 of file [interface.h](#).

### 3.8.2.27 check\_template

```
GtkCheckButton* Window::check_template[MAX_NINPUTS]
```

Array of GtkCheckButtons to set the input templates.

Definition at line 213 of file [interface.h](#).

### 3.8.2.28 combo\_experiment

```
GtkComboBoxText* Window::combo_experiment
```

[Experiment](#) GtkComboBoxEntry.

Definition at line 205 of file [interface.h](#).

### 3.8.2.29 combo\_variable

GtkComboBoxText\* Window::combo\_variable

GtkComboBoxEntry to select a variable.

Definition at line 176 of file [interface.h](#).

### 3.8.2.30 entry\_result

GtkEntry\* Window::entry\_result

Result file GtkEntry.

Definition at line 98 of file [interface.h](#).

### 3.8.2.31 entry\_variable

GtkEntry\* Window::entry\_variable

GtkEntry to set the variable name.

Definition at line 181 of file [interface.h](#).

### 3.8.2.32 entry\_variables

GtkEntry\* Window::entry\_variables

Variables file GtkEntry.

Definition at line 100 of file [interface.h](#).

### 3.8.2.33 experiment

[Experiment](#)\* Window::experiment

Array of experiments data.

Definition at line 218 of file [interface.h](#).

### 3.8.2.34 frame\_algorithm

GtkFrame\* Window::frame\_algorithm

GtkFrame to set the algorithm.

Definition at line 114 of file [interface.h](#).

### 3.8.2.35 frame\_experiment

GtkFrame\* Window::frame\_experiment

[Experiment](#) GtkFrame.

Definition at line 203 of file [interface.h](#).

### 3.8.2.36 frame\_norm

GtkFrame\* Window::frame\_norm

GtkFrame to set the error norm.

Definition at line 101 of file [interface.h](#).

### 3.8.2.37 frame\_variable

GtkFrame\* Window::frame\_variable

[Variable](#) GtkFrame.

Definition at line 174 of file [interface.h](#).

### 3.8.2.38 grid

GtkGrid\* Window::grid

Main GtkGrid.

Definition at line 81 of file [interface.h](#).

### 3.8.2.39 grid\_algorithm

GtkGrid\* Window::grid\_algorithm

GtkGrid to set the algorithm.

Definition at line 115 of file [interface.h](#).

### 3.8.2.40 grid\_climbing

GtkGrid\* Window::grid\_climbing

GtkGrid to pack the hill climbing method widgets.

Definition at line 149 of file [interface.h](#).

#### 3.8.2.41 grid\_experiment

```
GtkGrid* Window::grid_experiment
```

[Experiment](#) GtkGrid.

Definition at line 204 of file [interface.h](#).

#### 3.8.2.42 grid\_files

```
GtkGrid* Window::grid_files
```

Files GtkGrid.

Definition at line 90 of file [interface.h](#).

#### 3.8.2.43 grid\_norm

```
GtkGrid* Window::grid_norm
```

GtkGrid to set the error norm.

Definition at line 102 of file [interface.h](#).

#### 3.8.2.44 grid\_variable

```
GtkGrid* Window::grid_variable
```

[Variable](#) GtkGrid.

Definition at line 175 of file [interface.h](#).

#### 3.8.2.45 id\_experiment

```
gulong Window::id_experiment
```

Identifier of the combo\_experiment signal.

Definition at line 221 of file [interface.h](#).

#### 3.8.2.46 id\_experiment\_name

```
gulong Window::id_experiment_name
```

Identifier of the button\_experiment signal.

Definition at line 222 of file [interface.h](#).

#### 3.8.2.47 id\_input

```
gulong Window::id_input[MAX_NINPUTS]
```

Array of identifiers of the button\_template signal.

Definition at line 227 of file [interface.h](#).

#### 3.8.2.48 id\_template

```
gulong Window::id_template[MAX_NINPUTS]
```

Array of identifiers of the check\_template signal.

Definition at line 225 of file [interface.h](#).

#### 3.8.2.49 id\_variable

```
gulong Window::id_variable
```

Identifier of the combo\_variable signal.

Definition at line 223 of file [interface.h](#).

#### 3.8.2.50 id\_variable\_label

```
gulong Window::id_variable_label
```

Identifier of the entry\_variable signal.

Definition at line 224 of file [interface.h](#).

#### 3.8.2.51 label\_adaptation

```
GtkLabel* Window::label_adaptation
```

GtkLabel to set the adaptation ratio.

Definition at line 144 of file [interface.h](#).

#### 3.8.2.52 label\_best

```
GtkLabel* Window::label_best
```

GtkLabel to set the best number.

Definition at line 131 of file [interface.h](#).

### 3.8.2.53 label\_bits

GtkLabel\* Window::label\_bits

Bits number GtkLabel.

Definition at line 198 of file [interface.h](#).

### 3.8.2.54 label\_estimates

GtkLabel\* Window::label\_estimates

GtkLabel to set the estimates number.

Definition at line 163 of file [interface.h](#).

### 3.8.2.55 label\_experiment

GtkLabel\* Window::label\_experiment

[Experiment](#) GtkLabel.

Definition at line 208 of file [interface.h](#).

### 3.8.2.56 label\_final\_steps

GtkLabel\* Window::label\_final\_steps

GtkLabel to set the final steps number.

Definition at line 160 of file [interface.h](#).

### 3.8.2.57 label\_generations

GtkLabel\* Window::label\_generations

GtkLabel to set the generations number.

Definition at line 136 of file [interface.h](#).

### 3.8.2.58 label\_iterations

GtkLabel\* Window::label\_iterations

GtkLabel to set the iterations number.

Definition at line 126 of file [interface.h](#).

### 3.8.2.59 label\_max

```
GtkLabel* Window::label_max
```

Maximum GtkLabel.

Definition at line 185 of file [interface.h](#).

### 3.8.2.60 label\_min

```
GtkLabel* Window::label_min
```

Minimum GtkLabel.

Definition at line 182 of file [interface.h](#).

### 3.8.2.61 label\_mutation

```
GtkLabel* Window::label_mutation
```

GtkLabel to set the mutation ratio.

Definition at line 139 of file [interface.h](#).

### 3.8.2.62 label\_p

```
GtkLabel* Window::label_p
```

GtkLabel to set the p parameter.

Definition at line 110 of file [interface.h](#).

### 3.8.2.63 label\_population

```
GtkLabel* Window::label_population
```

GtkLabel to set the population number.

Definition at line 133 of file [interface.h](#).

### 3.8.2.64 label\_precision

```
GtkLabel* Window::label_precision
```

Precision GtkLabel.

Definition at line 194 of file [interface.h](#).

### 3.8.2.65 label\_relaxation

```
GtkLabel* Window::label_relaxation
```

GtkLabel to set the relaxation parameter.

Definition at line 166 of file [interface.h](#).

### 3.8.2.66 label\_reproduction

```
GtkLabel* Window::label_reproduction
```

GtkLabel to set the reproduction ratio.

Definition at line 141 of file [interface.h](#).

### 3.8.2.67 label\_result

```
GtkLabel* Window::label_result
```

Result file GtkLabel.

Definition at line 97 of file [interface.h](#).

### 3.8.2.68 label\_simulations

```
GtkLabel* Window::label_simulations
```

GtkLabel to set the simulations number.

Definition at line 123 of file [interface.h](#).

### 3.8.2.69 label\_simulator

```
GtkLabel* Window::label_simulator
```

Simulator program GtkLabel.

Definition at line 91 of file [interface.h](#).

### 3.8.2.70 label\_step

```
GtkLabel* Window::label_step
```

GtkLabel to set the step.

Definition at line 200 of file [interface.h](#).



### 3.8.2.71 label\_steps

```
GtkLabel* Window::label_steps
```

GtkLabel to set the steps number.

Definition at line 158 of file [interface.h](#).

### 3.8.2.72 label\_sweeps

```
GtkLabel* Window::label_sweeps
```

Sweeps number GtkLabel.

Definition at line 196 of file [interface.h](#).

### 3.8.2.73 label\_threshold

```
GtkLabel* Window::label_threshold
```

GtkLabel to set the threshold.

Definition at line 170 of file [interface.h](#).

### 3.8.2.74 label\_tolerance

```
GtkLabel* Window::label_tolerance
```

GtkLabel to set the tolerance.

Definition at line 129 of file [interface.h](#).

### 3.8.2.75 label\_variable

```
GtkLabel* Window::label_variable
```

[Variable](#) GtkLabel.

Definition at line 180 of file [interface.h](#).

### 3.8.2.76 label\_variables

```
GtkLabel* Window::label_variables
```

Variables file GtkLabel.

Definition at line 99 of file [interface.h](#).

### 3.8.2.77 label\_weight

```
GtkLabel* Window::label_weight
```

Weight GtkLabel.

Definition at line 211 of file [interface.h](#).

### 3.8.2.78 logo

```
GdkPixbuf* Window::logo
```

Logo GdkPixbuf.

Definition at line 217 of file [interface.h](#).

### 3.8.2.79 nexperiments

```
unsigned int Window::nexperiments
```

Number of experiments.

Definition at line 229 of file [interface.h](#).

### 3.8.2.80 nvariables

```
unsigned int Window::nvariables
```

Number of variables.

Definition at line 230 of file [interface.h](#).

### 3.8.2.81 scrolled\_max

```
GtkScrolledWindow* Window::scrolled_max
```

Maximum GtkScrolledWindow.

Definition at line 187 of file [interface.h](#).

### 3.8.2.82 scrolled\_maxabs

```
GtkScrolledWindow* Window::scrolled_maxabs
```

Absolute maximum GtkScrolledWindow.

Definition at line 193 of file [interface.h](#).

### 3.8.2.83 scrolled\_min

```
GtkScrolledWindow* Window::scrolled_min
```

Minimum GtkScrolledWindow.

Definition at line 184 of file [interface.h](#).

### 3.8.2.84 scrolled\_minabs

```
GtkScrolledWindow* Window::scrolled_minabs
```

Absolute minimum GtkScrolledWindow.

Definition at line 190 of file [interface.h](#).

### 3.8.2.85 scrolled\_p

```
GtkScrolledWindow* Window::scrolled_p
```

GtkScrolledWindow to set the p parameter.

Definition at line 112 of file [interface.h](#).

### 3.8.2.86 scrolled\_step

```
GtkScrolledWindow* Window::scrolled_step
```

step GtkScrolledWindow.

Definition at line 202 of file [interface.h](#).

### 3.8.2.87 scrolled\_threshold

```
GtkScrolledWindow* Window::scrolled_threshold
```

GtkScrolledWindow to set the threshold.

Definition at line 172 of file [interface.h](#).

### 3.8.2.88 spin\_adaptation

```
GtkSpinButton* Window::spin_adaptation
```

GtkSpinButton to set the adaptation ratio.

Definition at line 145 of file [interface.h](#).

### 3.8.2.89 spin\_bests

GtkSpinButton\* Window::spin\_bests

GtkSpinButton to set the best number.

Definition at line 132 of file [interface.h](#).

### 3.8.2.90 spin\_bits

GtkSpinButton\* Window::spin\_bits

Bits number GtkSpinButton.

Definition at line 199 of file [interface.h](#).

### 3.8.2.91 spin\_estimates

GtkSpinButton\* Window::spin\_estimates

GtkSpinButton to set the estimates number.

Definition at line 164 of file [interface.h](#).

### 3.8.2.92 spin\_final\_steps

GtkSpinButton\* Window::spin\_final\_steps

GtkSpinButton to set the final steps number.

Definition at line 161 of file [interface.h](#).

### 3.8.2.93 spin\_generations

GtkSpinButton\* Window::spin\_generations

GtkSpinButton to set the generations number.

Definition at line 137 of file [interface.h](#).

### 3.8.2.94 spin\_iterations

GtkSpinButton\* Window::spin\_iterations

GtkSpinButton to set the iterations number.

Definition at line 127 of file [interface.h](#).

### 3.8.2.95 spin\_max

GtkSpinButton\* Window::spin\_max

Maximum GtkSpinButton.

Definition at line 186 of file [interface.h](#).

### 3.8.2.96 spin\_maxabs

GtkSpinButton\* Window::spin\_maxabs

Absolute maximum GtkSpinButton.

Definition at line 192 of file [interface.h](#).

### 3.8.2.97 spin\_min

GtkSpinButton\* Window::spin\_min

Minimum GtkSpinButton.

Definition at line 183 of file [interface.h](#).

### 3.8.2.98 spin\_minabs

GtkSpinButton\* Window::spin\_minabs

Absolute minimum GtkSpinButton.

Definition at line 189 of file [interface.h](#).

### 3.8.2.99 spin\_mutation

GtkSpinButton\* Window::spin\_mutation

GtkSpinButton to set the mutation ratio.

Definition at line 140 of file [interface.h](#).

### 3.8.2.100 spin\_p

GtkSpinButton\* Window::spin\_p

GtkSpinButton to set the p parameter.

Definition at line 111 of file [interface.h](#).

### 3.8.2.101 spin\_population

```
GtkSpinButton* Window::spin_population
```

GtkSpinButton to set the population number.

Definition at line 134 of file [interface.h](#).

### 3.8.2.102 spin\_precision

```
GtkSpinButton* Window::spin_precision
```

Precision digits GtkSpinButton.

Definition at line 195 of file [interface.h](#).

### 3.8.2.103 spin\_relaxation

```
GtkSpinButton* Window::spin_relaxation
```

GtkSpinButton to set the relaxation parameter.

Definition at line 168 of file [interface.h](#).

### 3.8.2.104 spin\_reproduction

```
GtkSpinButton* Window::spin_reproduction
```

GtkSpinButton to set the reproduction ratio.

Definition at line 142 of file [interface.h](#).

### 3.8.2.105 spin\_simulations

```
GtkSpinButton* Window::spin_simulations
```

GtkSpinButton to set the simulations number.

Definition at line 124 of file [interface.h](#).

### 3.8.2.106 spin\_step

```
GtkSpinButton* Window::spin_step
```

GtkSpinButton to set the step.

Definition at line 201 of file [interface.h](#).

### 3.8.2.107 spin\_steps

GtkSpinButton\* Window::spin\_steps

GtkSpinButton to set the steps number.

Definition at line 159 of file [interface.h](#).

### 3.8.2.108 spin\_sweeps

GtkSpinButton\* Window::spin\_sweeps

Sweeps number GtkSpinButton.

Definition at line 197 of file [interface.h](#).

### 3.8.2.109 spin\_threshold

GtkSpinButton\* Window::spin\_threshold

GtkSpinButton to set the threshold.

Definition at line 171 of file [interface.h](#).

### 3.8.2.110 spin\_tolerance

GtkSpinButton\* Window::spin\_tolerance

GtkSpinButton to set the tolerance.

Definition at line 130 of file [interface.h](#).

### 3.8.2.111 spin\_weight

GtkSpinButton\* Window::spin\_weight

Weight GtkSpinButton.

Definition at line 212 of file [interface.h](#).

### 3.8.2.112 variable

Variable\* Window::variable

Array of variables data.

Definition at line 219 of file [interface.h](#).

### 3.8.2.113 window

GtkWindow\* Window::window

Main GtkWindow.

Definition at line 80 of file [interface.h](#).

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

- [interface.h](#)





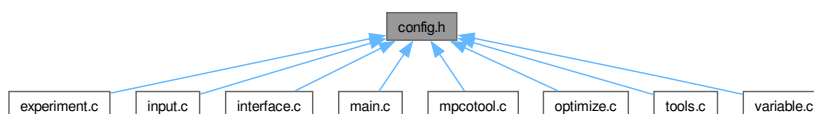
## Chapter 4

# File Documentation

### 4.1 config.h File Reference

Configuration header file.

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



#### Macros

- `#define MAX_NINPUTS 8`  
*Maximum number of input files in the simulator program.*
- `#define NALGORITHMMS 4`  
*Number of stochastic algorithms.*
- `#define NCLIMBINGS 2`  
*Number of hill climbing estimate methods.*
- `#define NNORMS 4`  
*Number of error norms.*
- `#define NPRECISIONS 15`  
*Number of precisions.*
- `#define DEFAULT_PRECISION (NPRECISIONS - 1)`  
*Default precision digits.*
- `#define DEFAULT_RANDOM_SEED 7007`  
*Default pseudo-random numbers seed.*
- `#define DEFAULT_RELAXATION 1.`  
*Default relaxation parameter.*
- `#define LOCALE_DIR "locales"`  
*Locales directory.*

- #define PROGRAM\_INTERFACE "mpcotool"  
*Name of the interface program.*
- #define LABEL\_ABSOLUTE\_MINIMUM "absolute\_minimum"  
*absolute minimum label.*
- #define LABEL\_ABSOLUTE\_MAXIMUM "absolute\_maximum"  
*absolute maximum label.*
- #define LABEL\_ADAPTATION "adaptation"  
*adaption label.*
- #define LABEL\_ALGORITHM "algorithm"  
*algoritm label.*
- #define LABEL\_CLEANER "cleaner"  
*cleaner label.*
- #define LABEL\_CLIMBING "climbing"  
*climbing label.*
- #define LABEL\_COORDINATES "coordinates"  
*coordinates label.*
- #define LABEL\_EUCLIDIAN "euclidian"  
*euclidian label.*
- #define LABEL\_EVALUATOR "evaluator"  
*evaluator label.*
- #define LABEL\_EXPERIMENT "experiment"  
*experiment label.*
- #define LABEL\_EXPERIMENTS "experiments"  
*experiment label.*
- #define LABEL\_GENETIC "genetic"  
*genetic label.*
- #define LABEL\_INPUT1 "input1"  
*input1 label.*
- #define LABEL\_INPUT2 "input2"  
*input2 label.*
- #define LABEL\_INPUT3 "input3"  
*input3 label.*
- #define LABEL\_INPUT4 "input4"  
*input4 label.*
- #define LABEL\_INPUT5 "input5"  
*input5 label.*
- #define LABEL\_INPUT6 "input6"  
*input6 label.*
- #define LABEL\_INPUT7 "input7"  
*input7 label.*
- #define LABEL\_INPUT8 "input8"  
*input8 label.*
- #define LABEL\_MINIMUM "minimum"  
*minimum label.*
- #define LABEL\_MAXIMUM "maximum"  
*maximum label.*
- #define LABEL\_MONTE\_CARLO "Monte-Carlo"  
*Monte-Carlo label.*
- #define LABEL\_MUTATION "mutation"  
*mutation label.*
- #define LABEL\_NAME "name"

- name label.*
- #define LABEL\_NBEST "nbest"  
*nbest label.*
- #define LABEL\_NBITS "nbits"  
*nbits label.*
- #define LABEL\_NESTIMATES "nestimates"  
*nestimates label.*
- #define LABEL\_NFINAL\_STEPS "nfinal\_steps"  
*nfinal\_steps label.*
- #define LABEL\_NGENERATIONS "ngenerations"  
*ngenerations label.*
- #define LABEL\_NITERATIONS "niterations"  
*niterations label.*
- #define LABEL\_NORM "norm"  
*norm label.*
- #define LABEL\_NPOPULATION "npopulation"  
*npopulation label.*
- #define LABEL\_NSIMULATIONS "nsimulations"  
*nsimulations label.*
- #define LABEL\_NSTEPS "nsteps"  
*nsteps label.*
- #define LABEL\_NSWEEPS "nsweeps"  
*nsweeps label.*
- #define LABEL\_OPTIMIZE "optimize"  
*optimize label.*
- #define LABEL\_ORTHOGONAL "orthogonal"  
*orthogonal label.*
- #define LABEL\_P "p"  
*p label.*
- #define LABEL\_PRECISION "precision"  
*precision label.*
- #define LABEL\_RANDOM "random"  
*random label.*
- #define LABEL\_RELAXATION "relaxation"  
*relaxation label.*
- #define LABEL\_REPRODUCTION "reproduction"  
*reproduction label.*
- #define LABEL\_RESULT\_FILE "result\_file"  
*result\_file label.*
- #define LABEL\_SIMULATOR "simulator"  
*simulator label.*
- #define LABEL\_SEED "seed"  
*seed label.*
- #define LABEL\_STEP "step"  
*step label.*
- #define LABEL\_SWEEP "sweep"  
*sweep label.*
- #define LABEL\_TAXICAB "taxicab"  
*taxicab label.*
- #define LABEL\_TEMPLATE1 "template1"  
*template1 label.*

- `#define LABEL_TEMPLATE2 "template2"`  
*template2 label.*
- `#define LABEL_TEMPLATE3 "template3"`  
*template3 label.*
- `#define LABEL_TEMPLATE4 "template4"`  
*template4 label.*
- `#define LABEL_TEMPLATE5 "template5"`  
*template5 label.*
- `#define LABEL_TEMPLATE6 "template6"`  
*template6 label.*
- `#define LABEL_TEMPLATE7 "template7"`  
*template7 label.*
- `#define LABEL_TEMPLATE8 "template8"`  
*template8 label.*
- `#define LABEL_THRESHOLD "threshold"`  
*threshold label.*
- `#define LABEL_TOLERANCE "tolerance"`  
*tolerance label.*
- `#define LABEL_VARIABLE "variable"`  
*variable label.*
- `#define LABEL_VARIABLES "variables"`  
*variables label.*
- `#define LABEL_VARIABLES_FILE "variables_file"`  
*variables label.*
- `#define LABEL_WEIGHT "weight"`  
*weight label.*

## Enumerations

- `enum INPUT_TYPE { INPUT_TYPE_XML = 0 , INPUT_TYPE_JSON = 1 }`  
*Enum to define the input file types.*

### 4.1.1 Detailed Description

Configuration header file.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2018, all rights reserved.

Definition in file [config.h](#).

## 4.1.2 Macro Definition Documentation

### 4.1.2.1 DEFAULT\_PRECISION

```
#define DEFAULT_PRECISION (NPRECISIONS - 1)
```

Default precision digits.

Definition at line 55 of file [config.h](#).

### 4.1.2.2 DEFAULT\_RANDOM\_SEED

```
#define DEFAULT_RANDOM_SEED 7007
```

Default pseudo-random numbers seed.

Definition at line 56 of file [config.h](#).

### 4.1.2.3 DEFAULT\_RELAXATION

```
#define DEFAULT_RELAXATION 1.
```

Default relaxation parameter.

Definition at line 57 of file [config.h](#).

### 4.1.2.4 LABEL\_ABSOLUTE\_MAXIMUM

```
#define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
```

absolute maximum label.

Definition at line 69 of file [config.h](#).

### 4.1.2.5 LABEL\_ABSOLUTE\_MINIMUM

```
#define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
```

absolute minimum label.

Definition at line 67 of file [config.h](#).

### 4.1.2.6 LABEL\_ADAPTATION

```
#define LABEL_ADAPTATION "adaptation"
```

adaption label.

Definition at line 70 of file [config.h](#).

#### 4.1.2.7 LABEL\_ALGORITHM

```
#define LABEL_ALGORITHM "algorithm"
```

algorithm label.

Definition at line 71 of file [config.h](#).

#### 4.1.2.8 LABEL\_CLEANER

```
#define LABEL_CLEANER "cleaner"
```

cleaner label.

Definition at line 72 of file [config.h](#).

#### 4.1.2.9 LABEL\_CLIMBING

```
#define LABEL_CLIMBING "climbing"
```

climbing label.

Definition at line 73 of file [config.h](#).

#### 4.1.2.10 LABEL\_COORDINATES

```
#define LABEL_COORDINATES "coordinates"
```

coordinates label.

Definition at line 74 of file [config.h](#).

#### 4.1.2.11 LABEL\_EUCLIDIAN

```
#define LABEL_EUCLIDIAN "euclidian"
```

euclidian label.

Definition at line 75 of file [config.h](#).

#### 4.1.2.12 LABEL\_EVALUATOR

```
#define LABEL_EVALUATOR "evaluator"
```

evaluator label.

Definition at line 76 of file [config.h](#).

#### 4.1.2.13 LABEL\_EXPERIMENT

```
#define LABEL_EXPERIMENT "experiment"
```

experiment label.

Definition at line 77 of file [config.h](#).

#### 4.1.2.14 LABEL\_EXPERIMENTS

```
#define LABEL_EXPERIMENTS "experiments"
```

experiment label.

Definition at line 78 of file [config.h](#).

#### 4.1.2.15 LABEL\_GENETIC

```
#define LABEL_GENETIC "genetic"
```

genetic label.

Definition at line 79 of file [config.h](#).

#### 4.1.2.16 LABEL\_INPUT1

```
#define LABEL_INPUT1 "input1"
```

input1 label.

Definition at line 80 of file [config.h](#).

#### 4.1.2.17 LABEL\_INPUT2

```
#define LABEL_INPUT2 "input2"
```

input2 label.

Definition at line 81 of file [config.h](#).

#### 4.1.2.18 LABEL\_INPUT3

```
#define LABEL_INPUT3 "input3"
```

input3 label.

Definition at line 82 of file [config.h](#).

#### 4.1.2.19 LABEL\_INPUT4

```
#define LABEL_INPUT4 "input4"
```

input4 label.

Definition at line 83 of file [config.h](#).

#### 4.1.2.20 LABEL\_INPUT5

```
#define LABEL_INPUT5 "input5"
```

input5 label.

Definition at line 84 of file [config.h](#).

#### 4.1.2.21 LABEL\_INPUT6

```
#define LABEL_INPUT6 "input6"
```

input6 label.

Definition at line 85 of file [config.h](#).

#### 4.1.2.22 LABEL\_INPUT7

```
#define LABEL_INPUT7 "input7"
```

input7 label.

Definition at line 86 of file [config.h](#).

#### 4.1.2.23 LABEL\_INPUT8

```
#define LABEL_INPUT8 "input8"
```

input8 label.

Definition at line 87 of file [config.h](#).

#### 4.1.2.24 LABEL\_MAXIMUM

```
#define LABEL_MAXIMUM "maximum"
```

maximum label.

Definition at line 89 of file [config.h](#).



#### 4.1.2.25 LABEL\_MINIMUM

```
#define LABEL_MINIMUM "minimum"
```

minimum label.

Definition at line 88 of file [config.h](#).

#### 4.1.2.26 LABEL\_MONTE\_CARLO

```
#define LABEL_MONTE_CARLO "Monte-Carlo"
```

Monte-Carlo label.

Definition at line 90 of file [config.h](#).

#### 4.1.2.27 LABEL\_MUTATION

```
#define LABEL_MUTATION "mutation"
```

mutation label.

Definition at line 91 of file [config.h](#).

#### 4.1.2.28 LABEL\_NAME

```
#define LABEL_NAME "name"
```

name label.

Definition at line 92 of file [config.h](#).

#### 4.1.2.29 LABEL\_NBEST

```
#define LABEL_NBEST "nbest"
```

nbest label.

Definition at line 93 of file [config.h](#).

#### 4.1.2.30 LABEL\_NBITS

```
#define LABEL_NBITS "nbits"
```

nbits label.

Definition at line 94 of file [config.h](#).

#### 4.1.2.31 LABEL\_NESTIMATES

```
#define LABEL_NESTIMATES "nestimates"
```

nestimates label.

Definition at line 95 of file [config.h](#).

#### 4.1.2.32 LABEL\_NFINAL\_STEPS

```
#define LABEL_NFINAL_STEPS "nfinal_steps"
```

nfinal\_steps label.

Definition at line 96 of file [config.h](#).

#### 4.1.2.33 LABEL\_NGENERATIONS

```
#define LABEL_NGENERATIONS "ngenerations"
```

ngenerations label.

Definition at line 97 of file [config.h](#).

#### 4.1.2.34 LABEL\_NITERATIONS

```
#define LABEL_NITERATIONS "niterations"
```

niterations label.

Definition at line 98 of file [config.h](#).

#### 4.1.2.35 LABEL\_NORM

```
#define LABEL_NORM "norm"
```

norm label.

Definition at line 99 of file [config.h](#).

#### 4.1.2.36 LABEL\_NPOPULATION

```
#define LABEL_NPOPULATION "npopulation"
```

npopulation label.

Definition at line 100 of file [config.h](#).

#### 4.1.2.37 LABEL\_NSIMULATIONS

```
#define LABEL_NSIMULATIONS "nsimulations"
```

nsimulations label.

Definition at line 101 of file [config.h](#).

#### 4.1.2.38 LABEL\_NSTEPS

```
#define LABEL_NSTEPS "nsteps"
```

nsteps label.

Definition at line 102 of file [config.h](#).

#### 4.1.2.39 LABEL\_NSWEEPS

```
#define LABEL_NSWEEPS "nsweeps"
```

nsweeps label.

Definition at line 103 of file [config.h](#).

#### 4.1.2.40 LABEL\_OPTIMIZE

```
#define LABEL_OPTIMIZE "optimize"
```

optimize label.

Definition at line 104 of file [config.h](#).

#### 4.1.2.41 LABEL\_ORTHOGONAL

```
#define LABEL_ORTHOGONAL "orthogonal"
```

orthogonal label.

Definition at line 105 of file [config.h](#).

#### 4.1.2.42 LABEL\_P

```
#define LABEL_P "p"
```

p label.

Definition at line 106 of file [config.h](#).

#### 4.1.2.43 LABEL\_PRECISION

```
#define LABEL_PRECISION "precision"
```

precision label.

Definition at line 107 of file [config.h](#).

#### 4.1.2.44 LABEL\_RANDOM

```
#define LABEL_RANDOM "random"
```

random label.

Definition at line 108 of file [config.h](#).

#### 4.1.2.45 LABEL\_RELAXATION

```
#define LABEL_RELAXATION "relaxation"
```

relaxation label.

Definition at line 109 of file [config.h](#).

#### 4.1.2.46 LABEL\_REPRODUCTION

```
#define LABEL_REPRODUCTION "reproduction"
```

reproduction label.

Definition at line 110 of file [config.h](#).

#### 4.1.2.47 LABEL\_RESULT\_FILE

```
#define LABEL_RESULT_FILE "result_file"
```

result\_file label.

Definition at line 111 of file [config.h](#).

#### 4.1.2.48 LABEL\_SEED

```
#define LABEL_SEED "seed"
```

seed label.

Definition at line 113 of file [config.h](#).

#### 4.1.2.49 LABEL\_SIMULATOR

```
#define LABEL_SIMULATOR "simulator"
```

simulator label.

Definition at line 112 of file [config.h](#).

#### 4.1.2.50 LABEL\_STEP

```
#define LABEL_STEP "step"
```

step label.

Definition at line 114 of file [config.h](#).

#### 4.1.2.51 LABEL\_SWEEP

```
#define LABEL_SWEEP "sweep"
```

sweep label.

Definition at line 115 of file [config.h](#).

#### 4.1.2.52 LABEL\_TAXICAB

```
#define LABEL_TAXICAB "taxicab"
```

taxicab label.

Definition at line 116 of file [config.h](#).

#### 4.1.2.53 LABEL\_TEMPLATE1

```
#define LABEL_TEMPLATE1 "template1"
```

template1 label.

Definition at line 117 of file [config.h](#).

#### 4.1.2.54 LABEL\_TEMPLATE2

```
#define LABEL_TEMPLATE2 "template2"
```

template2 label.

Definition at line 118 of file [config.h](#).

#### 4.1.2.55 LABEL\_TEMPLATE3

```
#define LABEL_TEMPLATE3 "template3"
```

template3 label.

Definition at line 119 of file [config.h](#).

#### 4.1.2.56 LABEL\_TEMPLATE4

```
#define LABEL_TEMPLATE4 "template4"
```

template4 label.

Definition at line 120 of file [config.h](#).

#### 4.1.2.57 LABEL\_TEMPLATE5

```
#define LABEL_TEMPLATE5 "template5"
```

template5 label.

Definition at line 121 of file [config.h](#).

#### 4.1.2.58 LABEL\_TEMPLATE6

```
#define LABEL_TEMPLATE6 "template6"
```

template6 label.

Definition at line 122 of file [config.h](#).

#### 4.1.2.59 LABEL\_TEMPLATE7

```
#define LABEL_TEMPLATE7 "template7"
```

template7 label.

Definition at line 123 of file [config.h](#).

#### 4.1.2.60 LABEL\_TEMPLATE8

```
#define LABEL_TEMPLATE8 "template8"
```

template8 label.

Definition at line 124 of file [config.h](#).

#### 4.1.2.61 LABEL\_THRESHOLD

```
#define LABEL_THRESHOLD "threshold"
```

threshold label.

Definition at line 125 of file [config.h](#).

#### 4.1.2.62 LABEL\_TOLERANCE

```
#define LABEL_TOLERANCE "tolerance"
```

tolerance label.

Definition at line 126 of file [config.h](#).

#### 4.1.2.63 LABEL\_VARIABLE

```
#define LABEL_VARIABLE "variable"
```

variable label.

Definition at line 127 of file [config.h](#).

#### 4.1.2.64 LABEL\_VARIABLES

```
#define LABEL_VARIABLES "variables"
```

variables label.

Definition at line 128 of file [config.h](#).

#### 4.1.2.65 LABEL\_VARIABLES\_FILE

```
#define LABEL_VARIABLES_FILE "variables_file"
```

variables label.

Definition at line 129 of file [config.h](#).

#### 4.1.2.66 LABEL\_WEIGHT

```
#define LABEL_WEIGHT "weight"
```

weight label.

Definition at line 130 of file [config.h](#).

#### 4.1.2.67 LOCALE\_DIR

```
#define LOCALE_DIR "locales"
```

Locales directory.

Definition at line 61 of file [config.h](#).

#### 4.1.2.68 MAX\_NINPUTS

```
#define MAX_NINPUTS 8
```

Maximum number of input files in the simulator program.

Definition at line 47 of file [config.h](#).

#### 4.1.2.69 NALGORITHMS

```
#define NALGORITHMS 4
```

Number of stochastic algorithms.

Definition at line 48 of file [config.h](#).

#### 4.1.2.70 NCLIMBINGS

```
#define NCLIMBINGS 2
```

Number of hill climbing estimate methods.

Definition at line 49 of file [config.h](#).

#### 4.1.2.71 NNORMS

```
#define NNORMS 4
```

Number of error norms.

Definition at line 50 of file [config.h](#).

#### 4.1.2.72 NPRECISIONS

```
#define NPRECISIONS 15
```

Number of precisions.

Definition at line 51 of file [config.h](#).

#### 4.1.2.73 PROGRAM\_INTERFACE

```
#define PROGRAM_INTERFACE "mpcotool"
```

Name of the interface program.

Definition at line 62 of file [config.h](#).

### 4.1.3 Enumeration Type Documentation

#### 4.1.3.1 INPUT\_TYPE

```
enum INPUT_TYPE
```

Enum to define the input file types.



## Enumerator

INPUT_TYPE_XML	XML input file.
INPUT_TYPE_JSON	JSON input file.

Definition at line 135 of file [config.h](#).

```
00136 {
00137     INPUT_TYPE_XML = 0,
00138     INPUT_TYPE_JSON = 1
00139 };
```

## 4.2 config.h

[Go to the documentation of this file.](#)

```
00001 /* config.h. Generated from config.h.in by configure. */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2018, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014 1. Redistributions of source code must retain the above copyright notice,
00015    this list of conditions and the following disclaimer.
00016
00017 2. Redistributions in binary form must reproduce the above copyright notice,
00018    this list of conditions and the following disclaimer in the
00019    documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00033 #ifndef CONFIG__H
00034 #define CONFIG__H 1
00035
00036 /* #undef HAVE_MPI */
00037
00038 // Array sizes
00039
00040 #define MAX_NINPUTS 8
00041 #define NALGORITHMS 4
00042 #define NCLIMBINGS 2
00043 #define NNORMS 4
00044 #define NPRECISIONS 15
00045
00046 // Default choices
00047
00048 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00049 #define DEFAULT_RANDOM_SEED 7007
00050 #define DEFAULT_RELAXATION 1.
00051
00052 // Interface labels
00053
00054 #define LOCALE_DIR "locales"
00055 #define PROGRAM_INTERFACE "mpcotool"
00056
00057 // Labels
00058
00059 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00060 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00061 #define LABEL_ADAPTATION "adaptation"
00062 #define LABEL_ALGORITHM "algorithm"
```

```

00072 #define LABEL_CLEANER "cleaner"
00073 #define LABEL_CLIMBING "climbing"
00074 #define LABEL_COORDINATES "coordinates"
00075 #define LABEL_EUCLIDIAN "euclidian"
00076 #define LABEL_EVALUATOR "evaluator"
00077 #define LABEL_EXPERIMENT "experiment"
00078 #define LABEL_EXPERIMENTS "experiments"
00079 #define LABEL_GENETIC "genetic"
00080 #define LABEL_INPUT1 "input1"
00081 #define LABEL_INPUT2 "input2"
00082 #define LABEL_INPUT3 "input3"
00083 #define LABEL_INPUT4 "input4"
00084 #define LABEL_INPUT5 "input5"
00085 #define LABEL_INPUT6 "input6"
00086 #define LABEL_INPUT7 "input7"
00087 #define LABEL_INPUT8 "input8"
00088 #define LABEL_MINIMUM "minimum"
00089 #define LABEL_MAXIMUM "maximum"
00090 #define LABEL_MONTE_CARLO "Monte-Carlo"
00091 #define LABEL_MUTATION "mutation"
00092 #define LABEL_NAME "name"
00093 #define LABEL_NBEST "nbest"
00094 #define LABEL_NBITS "nbits"
00095 #define LABEL_NESTIMATES "nestimates"
00096 #define LABEL_NFINAL_STEPS "nfinal_steps"
00097 #define LABEL_NGENERATIONS "ngenerations"
00098 #define LABEL_NITERATIONS "niterations"
00099 #define LABEL_NORM "norm"
00100 #define LABEL_NPOPULATION "npopulation"
00101 #define LABEL_NSIMULATIONS "nsimulations"
00102 #define LABEL_NSTEPS "nsteps"
00103 #define LABEL_NSWEEPS "nsweeps"
00104 #define LABEL_OPTIMIZE "optimize"
00105 #define LABEL_ORTHOGONAL "orthogonal"
00106 #define LABEL_P "p"
00107 #define LABEL_PRECISION "precision"
00108 #define LABEL_RANDOM "random"
00109 #define LABEL_RELAXATION "relaxation"
00110 #define LABEL_REPRODUCTION "reproduction"
00111 #define LABEL_RESULT_FILE "result_file"
00112 #define LABEL_SIMULATOR "simulator"
00113 #define LABEL_SEED "seed"
00114 #define LABEL_STEP "step"
00115 #define LABEL_SWEEP "sweep"
00116 #define LABEL_TAXICAB "taxicab"
00117 #define LABEL_TEMPLATE1 "template1"
00118 #define LABEL_TEMPLATE2 "template2"
00119 #define LABEL_TEMPLATE3 "template3"
00120 #define LABEL_TEMPLATE4 "template4"
00121 #define LABEL_TEMPLATE5 "template5"
00122 #define LABEL_TEMPLATE6 "template6"
00123 #define LABEL_TEMPLATE7 "template7"
00124 #define LABEL_TEMPLATE8 "template8"
00125 #define LABEL_THRESHOLD "threshold"
00126 #define LABEL_TOLERANCE "tolerance"
00127 #define LABEL_VARIABLE "variable"
00128 #define LABEL_VARIABLES "variables"
00129 #define LABEL_VARIABLES_FILE "variables_file"
00130 #define LABEL_WEIGHT "weight"
00131
00132 // Enumerations
00133
00134 enum INPUT_TYPE
00135 {
00136     INPUT_TYPE_XML = 0,
00137     INPUT_TYPE_JSON = 1
00138 };
00139
00140
00141 #endif

```

## 4.3 experiment.c File Reference

Source file to define the experiment data.

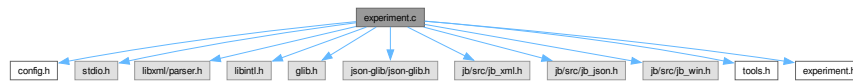
```

#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>

```

```
#include <glib.h>
#include <json-glib/json-glib.h>
#include "jb/src/jb_xml.h"
#include "jb/src/jb_json.h"
#include "jb/src/jb_win.h"
#include "tools.h"
#include "experiment.h"
```

Include dependency graph for experiment.c:



## Macros

- `#define DEBUG\_EXPERIMENT 0`  
Macro to debug experiment functions.

## Functions

- static void [experiment\\_new](#) ([Experiment](#) \*experiment)
- void [experiment\\_free](#) ([Experiment](#) \*experiment, unsigned int type)
- void [experiment\\_error](#) ([Experiment](#) \*experiment, char \*message)
- int [experiment\\_open\\_xml](#) ([Experiment](#) \*experiment, xmlNode \*node, unsigned int ninputs)
- int [experiment\\_open\\_json](#) ([Experiment](#) \*experiment, JsonNode \*node, unsigned int ninputs)

## Variables

- const char \* [stencil](#) [[MAX\\_NINPUTS](#)]  
Array of strings with stencil labels.
- const char \* [stencilbin](#) [[MAX\\_NINPUTS](#)]  
Array of strings with binary stencil labels.

### 4.3.1 Detailed Description

Source file to define the experiment data.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [experiment.c](#).

## 4.3.2 Macro Definition Documentation

### 4.3.2.1 DEBUG\_EXPERIMENT

```
#define DEBUG_EXPERIMENT 0
```

Macro to debug experiment functions.

Definition at line 51 of file [experiment.c](#).

## 4.3.3 Function Documentation

### 4.3.3.1 experiment\_error()

```
void experiment_error (
    Experiment * experiment,
    char * message )
```

Function to print a message error opening an [Experiment](#) struct.

#### Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>message</i>	Error message.

Definition at line 117 of file [experiment.c](#).

```
00119 {
00120     if (!experiment->name)
00121         error_message = g_strconcat (_, "Experiment", " ", message, NULL);
00122     else
00123         error_message = g_strconcat (_, "Experiment", " ", experiment->name, " ",
00124                                     message, NULL);
00125 }
```

### 4.3.3.2 experiment\_free()

```
void experiment_free (
    Experiment * experiment,
    unsigned int type )
```

Function to free the memory of an [Experiment](#) struct.

#### Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>type</i>	Type of input file.

Definition at line 88 of file [experiment.c](#).

```
00090 {
00091     unsigned int i;
00092     #if DEBUG_EXPERIMENT
00093         fprintf (stderr, "experiment_free: start\n");
00094     #endif
```

```

00095  if (type == INPUT_TYPE_XML)
00096  {
00097      for (i = 0; i < experiment->ninputs; ++i)
00098          xmlFree (experiment->stencil[i]);
00099      xmlFree (experiment->name);
00100  }
00101  else
00102  {
00103      for (i = 0; i < experiment->ninputs; ++i)
00104          g_free (experiment->stencil[i]);
00105      g_free (experiment->name);
00106  }
00107  experiment->ninputs = experiment->template_flags = 0;
00108  #if DEBUG_EXPERIMENT
00109      fprintf (stderr, "experiment_free: end\n");
00110  #endif
00111  }

```

#### 4.3.3.3 experiment\_new()

```

static void experiment_new (
    Experiment * experiment ) [static]

```

Function to create a new [Experiment](#) struct.

##### Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
-------------------	------------------------------------

Definition at line 69 of file [experiment.c](#).

```

00070 {
00071     unsigned int i;
00072     #if DEBUG_EXPERIMENT
00073         fprintf (stderr, "experiment_new: start\n");
00074     #endif
00075     experiment->name = NULL;
00076     experiment->ninputs = experiment->template_flags = 0;
00077     for (i = 0; i < MAX_NINPUTS; ++i)
00078         experiment->stencil[i] = NULL;
00079     #if DEBUG_EXPERIMENT
00080         fprintf (stderr, "input_new: end\n");
00081     #endif
00082 }

```

#### 4.3.3.4 experiment\_open\_json()

```

int experiment_open_json (
    Experiment * experiment,
    JsonNode * node,
    unsigned int ninputs )

```

Function to open the [Experiment](#) struct on a XML node.

##### Returns

1 on success, 0 on error.

##### Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>node</i>	JSON node.
<i>ninputs</i>	Number of the simulator input files.

Definition at line 270 of file `experiment.c`.

```

00274 {
00275     char buffer[64];
00276     JsonObject *object;
00277     const char *name;
00278     int error_code;
00279     unsigned int i;
00280     unsigned int flags = 1;
00281
00282     #if DEBUG_EXPERIMENT
00283         fprintf (stderr, "experiment_open_json: start\n");
00284     #endif
00285
00286     // Resetting experiment data
00287     experiment_new (experiment);
00288
00289     // Getting JSON object
00290     object = json_node_get_object (node);
00291
00292     // Reading the experimental data
00293     name = json_object_get_string_member (object, LABEL_NAME);
00294     if (!name)
00295     {
00296         experiment_error (experiment, _("no data file name"));
00297         goto exit_on_error;
00298     }
00299     experiment->name = g_strdup (name);
00300     #if DEBUG_EXPERIMENT
00301         fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00302     #endif
00303     experiment->weight
00304         = jb_json_object_get_float_with_default (object, LABEL_WEIGHT, &error_code,
00305                                                  1.);
00306     if (!error_code)
00307     {
00308         experiment_error (experiment, _("bad weight"));
00309         goto exit_on_error;
00310     }
00311     #if DEBUG_EXPERIMENT
00312         fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00313     #endif
00314     name = json_object_get_string_member (object, stencil[0]);
00315     if (name)
00316     {
00317         #if DEBUG_EXPERIMENT
00318             fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00319                     name, stencil[0]);
00320         #endif
00321         ++experiment->ninputs;
00322         experiment->template_flags |= flags;
00323     }
00324     else
00325     {
00326         name = json_object_get_string_member (object, stencilbin[0]);
00327         if (name)
00328         {
00329             #if DEBUG_EXPERIMENT
00330                 fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00331                         name, stencilbin[0]);
00332             #endif
00333             ++experiment->ninputs;
00334         }
00335         else
00336         {
00337             experiment_error (experiment, _("no template"));
00338             goto exit_on_error;
00339         }
00340     }
00341     experiment->stencil[0] = g_strdup (name);
00342     for (i = 1; i < MAX_NINPUTS; ++i)
00343     {
00344         #if DEBUG_EXPERIMENT
00345             fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00346         #endif
00347         flags <= 1;
00348         if (json_object_get_member (object, stencil[i]))
00349         {
00350             if (ninputs && ninputs <= i)
00351             {
00352                 experiment_error (experiment, _("bad templates number"));
00353                 goto exit_on_error;
00354             }
00355             name = json_object_get_string_member (object, stencil[i]);
00356             #if DEBUG_EXPERIMENT
00357                 fprintf (stderr,
00358                         "experiment_open_json: experiment=%s stencil%u=%s\n",
00359                         experiment->nexperiments, name, stencil[i]);

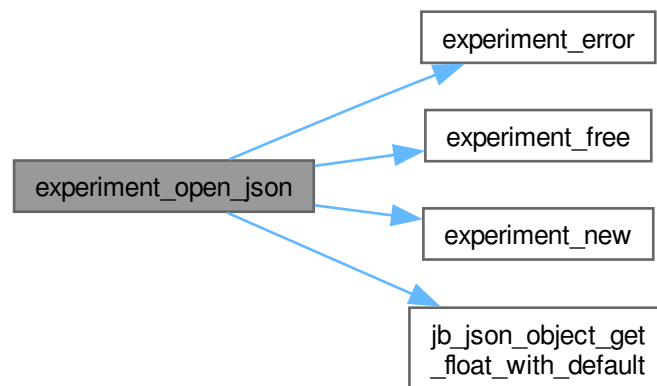
```

```

00360 #endif
00361     experiment->stencil[i] = g_strdup (name);
00362     ++experiment->ninputs;
00363     experiment->template_flags |= flags;
00364 }
00365 else if (json_object_get_member (object, stencilbin[i]))
00366 {
00367     if (ninputs && ninputs <= i)
00368     {
00369         experiment_error (experiment, _("bad templates number"));
00370         goto exit_on_error;
00371     }
00372     name = json_object_get_string_member (object, stencilbin[i]);
00373 #if DEBUG_EXPERIMENT
00374     fprintf (stderr,
00375             "experiment_open_json: experiment=%s stencil%u=%s\n",
00376             experiment->nexperiments, name, stencilbin[i]);
00377 #endif
00378     experiment->stencil[i] = g_strdup (name);
00379     ++experiment->ninputs;
00380 }
00381 else if (ninputs && ninputs > i)
00382 {
00383     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00384     experiment_error (experiment, buffer);
00385     goto exit_on_error;
00386 }
00387 else
00388     break;
00389 }
00390
00391 #if DEBUG_EXPERIMENT
00392 fprintf (stderr, "experiment_open_json: end\n");
00393 #endif
00394 return 1;
00395
00396 exit_on_error:
00397     experiment_free (experiment, INPUT_TYPE_JSON);
00398 #if DEBUG_EXPERIMENT
00399 fprintf (stderr, "experiment_open_json: end\n");
00400 #endif
00401 return 0;
00402 }

```

Here is the call graph for this function:



#### 4.3.3.5 experiment\_open\_xml()

```

int experiment_open_xml (
    Experiment * experiment,

```

```
xmlNode * node,
unsigned int ninputs )
```

Function to open the [Experiment](#) struct on a XML node.

#### Returns

1 on success, 0 on error.

#### Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>node</i>	XML node.
<i>ninputs</i>	Number of the simulator input files.

Definition at line 133 of file [experiment.c](#).

```
00137 {
00138     char buffer[64];
00139     int error_code;
00140     unsigned int i;
00141     unsigned int flags = 1;
00142
00143     #if DEBUG_EXPERIMENT
00144         fprintf (stderr, "experiment_open_xml: start\n");
00145     #endif
00146
00147     // Resetting experiment data
00148     experiment_new (experiment);
00149
00150     // Reading the experimental data
00151     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00152     if (!experiment->name)
00153     {
00154         experiment_error (experiment, _("no data file name"));
00155         goto exit_on_error;
00156     }
00157     #if DEBUG_EXPERIMENT
00158         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00159     #endif
00160     experiment->weight
00161         = jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_WEIGHT,
00162                                             &error_code, 1.);
00163     if (!error_code)
00164     {
00165         experiment_error (experiment, _("bad weight"));
00166         goto exit_on_error;
00167     }
00168     #if DEBUG_EXPERIMENT
00169         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00170     #endif
00171     experiment->stencil[0]
00172         = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00173     if (experiment->stencil[0])
00174     {
00175         #if DEBUG_EXPERIMENT
00176             fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
00177                     experiment->name, stencil[0]);
00178         #endif
00179         ++experiment->ninputs;
00180         experiment->template_flags |= flags;
00181     }
00182     else
00183     {
00184         experiment->stencil[0]
00185             = (char *) xmlGetProp (node, (const xmlChar *) stencilbin[0]);
00186         if (experiment->stencil[0])
00187         {
00188             #if DEBUG_EXPERIMENT
00189                 fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
00190                         experiment->name, stencilbin[0]);
00191             #endif
00192             ++experiment->ninputs;
00193         }
00194     }
00195     {
00196         experiment_error (experiment, _("no template"));
```

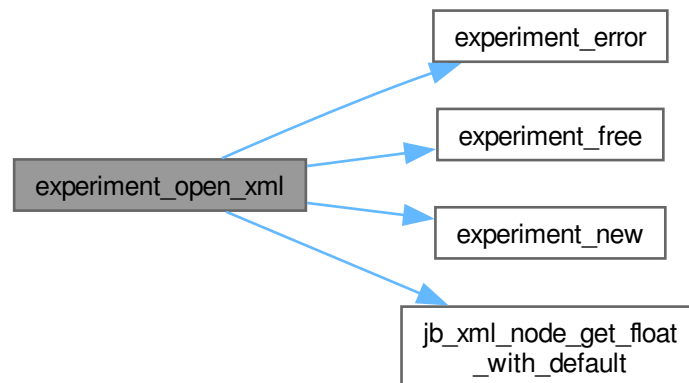


```

00197         goto exit_on_error;
00198     }
00199 }
00200 for (i = 1; i < MAX_NINPUTS; ++i)
00201 {
00202     #if DEBUG_EXPERIMENT
00203     fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00204     #endif
00205     flags <= 1;
00206     if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00207     {
00208         if (ninputs && ninputs <= i)
00209         {
00210             experiment_error (experiment, _("bad templates number"));
00211             goto exit_on_error;
00212         }
00213         experiment->stencil[i]
00214         = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00215     #if DEBUG_EXPERIMENT
00216     fprintf (stderr,
00217             "experiment_open_xml: experiment=%s stencil%u=%s\n",
00218             experiment->nexperiments, experiment->name,
00219             experiment->stencil[i]);
00220     #endif
00221     ++experiment->ninputs;
00222     experiment->template_flags |= flags;
00223 }
00224 else if (xmlHasProp (node, (const xmlChar *) stencilbin[i]))
00225 {
00226     if (ninputs && ninputs <= i)
00227     {
00228         experiment_error (experiment, _("bad templates number"));
00229         goto exit_on_error;
00230     }
00231     experiment->stencil[i]
00232     = (char *) xmlGetProp (node, (const xmlChar *) stencilbin[i]);
00233     #if DEBUG_EXPERIMENT
00234     fprintf (stderr,
00235             "experiment_open_xml: experiment=%s stencil%u=%s\n",
00236             experiment->nexperiments, experiment->name,
00237             experiment->stencil[i]);
00238     #endif
00239     ++experiment->ninputs;
00240 }
00241 else if (ninputs && ninputs > i)
00242 {
00243     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00244     experiment_error (experiment, buffer);
00245     goto exit_on_error;
00246 }
00247 else
00248     break;
00249 }
00250
00251 #if DEBUG_EXPERIMENT
00252 fprintf (stderr, "experiment_open_xml: end\n");
00253 #endif
00254 return 1;
00255
00256 exit_on_error:
00257 experiment_free (experiment, INPUT_TYPE_XML);
00258 #if DEBUG_EXPERIMENT
00259 fprintf (stderr, "experiment_open_xml: end\n");
00260 #endif
00261 return 0;
00262 }

```

Here is the call graph for this function:



### 4.3.4 Variable Documentation

#### 4.3.4.1 stencil

```
const char* stencil[MAX_NINPUTS]
```

**Initial value:**

```
= {
    LABEL_TEMPLATE1, LABEL_TEMPLATE2, LABEL_TEMPLATE3, LABEL_TEMPLATE4,
    LABEL_TEMPLATE5, LABEL_TEMPLATE6, LABEL_TEMPLATE7, LABEL_TEMPLATE8
}
```

Array of strings with stencil labels.

Definition at line 54 of file [experiment.c](#).

```
00054     {
00055     LABEL_TEMPLATE1, LABEL_TEMPLATE2, LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00056     LABEL_TEMPLATE5, LABEL_TEMPLATE6, LABEL_TEMPLATE7, LABEL_TEMPLATE8
00057     };
```

#### 4.3.4.2 stencilbin

```
const char* stencilbin[MAX_NINPUTS]
```

**Initial value:**

```
= {
    LABEL_INPUT1, LABEL_INPUT2, LABEL_INPUT3, LABEL_INPUT4,
    LABEL_INPUT5, LABEL_INPUT6, LABEL_INPUT7, LABEL_INPUT8
}
```

Array of strings with binary stencil labels.

Definition at line 60 of file [experiment.c](#).

```
00060     {
00061     LABEL_INPUT1, LABEL_INPUT2, LABEL_INPUT3, LABEL_INPUT4,
00062     LABEL_INPUT5, LABEL_INPUT6, LABEL_INPUT7, LABEL_INPUT8
00063     };
```

## 4.4 experiment.c

[Go to the documentation of this file.](#)

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032  #define _GNU_SOURCE
00033  #include "config.h"
00034  #include <stdio.h>
00035  #include <libxml/parser.h>
00036  #include <libintl.h>
00037  #include <glib.h>
00038  #include <json-glib/json-glib.h>
00039  #include "jb/src/jb_xml.h"
00040  #include "jb/src/jb_json.h"
00041  #include "jb/src/jb_win.h"
00042  #include "tools.h"
00043  #include "experiment.h"
00044
00045  #define DEBUG_EXPERIMENT 0
00046
00047  const char *stencil[MAX_NINPUTS] = {
00048      LABEL_TEMPLATE1, LABEL_TEMPLATE2, LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00049      LABEL_TEMPLATE5, LABEL_TEMPLATE6, LABEL_TEMPLATE7, LABEL_TEMPLATE8
00050  };
00051
00052  const char *stencilbin[MAX_NINPUTS] = {
00053      LABEL_INPUT1, LABEL_INPUT2, LABEL_INPUT3, LABEL_INPUT4,
00054      LABEL_INPUT5, LABEL_INPUT6, LABEL_INPUT7, LABEL_INPUT8
00055  };
00056
00057  static void
00058  experiment_new (Experiment * experiment)
00059  {
00060      unsigned int i;
00061      #if DEBUG_EXPERIMENT
00062          fprintf (stderr, "experiment_new: start\n");
00063      #endif
00064      experiment->name = NULL;
00065      experiment->ninputs = experiment->template_flags = 0;
00066      for (i = 0; i < MAX_NINPUTS; ++i)
00067          experiment->stencil[i] = NULL;
00068      #if DEBUG_EXPERIMENT
00069          fprintf (stderr, "input_new: end\n");
00070      #endif
00071  }
00072
00073  void
00074  experiment_free (Experiment * experiment,
00075                  unsigned int type)
00076  {
00077      unsigned int i;
00078      #if DEBUG_EXPERIMENT
00079          fprintf (stderr, "experiment_free: start\n");
00080      #endif
00081      if (type == INPUT_TYPE_XML)
00082          {

```

```

00097         for (i = 0; i < experiment->ninputs; ++i)
00098             xmlFree (experiment->stencil[i]);
00099         xmlFree (experiment->name);
00100     }
00101     else
00102     {
00103         for (i = 0; i < experiment->ninputs; ++i)
00104             g_free (experiment->stencil[i]);
00105         g_free (experiment->name);
00106     }
00107     experiment->ninputs = experiment->template_flags = 0;
00108     #if DEBUG_EXPERIMENT
00109     fprintf (stderr, "experiment_free: end\n");
00110     #endif
00111 }
00112
00113 void
00114 experiment_error (Experiment * experiment,
00115                  char *message)
00116 {
00117     if (!experiment->name)
00118         error_message = g_strconcat (_, "Experiment", ":", message, NULL);
00119     else
00120         error_message = g_strconcat (_, "Experiment", " ", experiment->name, ":",
00121                                     message, NULL);
00122 }
00123
00124 int
00125 experiment_open_xml (Experiment * experiment,
00126                    xmlNode * node,
00127                    unsigned int ninputs)
00128 {
00129     char buffer[64];
00130     int error_code;
00131     unsigned int i;
00132     unsigned int flags = 1;
00133
00134     #if DEBUG_EXPERIMENT
00135     fprintf (stderr, "experiment_open_xml: start\n");
00136     #endif
00137
00138     // Resetting experiment data
00139     experiment_new (experiment);
00140
00141     // Reading the experimental data
00142     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00143     if (!experiment->name)
00144     {
00145         experiment_error (experiment, _("no data file name"));
00146         goto exit_on_error;
00147     }
00148     #if DEBUG_EXPERIMENT
00149     fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00150     #endif
00151     experiment->weight
00152         = jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_WEIGHT,
00153                                              &error_code, 1.);
00154     if (!error_code)
00155     {
00156         experiment_error (experiment, _("bad weight"));
00157         goto exit_on_error;
00158     }
00159     #if DEBUG_EXPERIMENT
00160     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00161     #endif
00162     experiment->stencil[0]
00163         = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00164     if (experiment->stencil[0])
00165     {
00166         #if DEBUG_EXPERIMENT
00167         fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00168                 experiment->name, stencil[0]);
00169         #endif
00170         ++experiment->ninputs;
00171         experiment->template_flags |= flags;
00172     }
00173     else
00174     {
00175         experiment->stencil[0]
00176             = (char *) xmlGetProp (node, (const xmlChar *) stencilbin[0]);
00177         if (experiment->stencil[0])
00178         {
00179             #if DEBUG_EXPERIMENT
00180             fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00181                     experiment->name, stencilbin[0]);
00182             #endif
00183             ++experiment->ninputs;

```

```

00193     }
00194     else
00195     {
00196         experiment_error (experiment, _("no template"));
00197         goto exit_on_error;
00198     }
00199 }
00200 for (i = 1; i < MAX_NINPUTS; ++i)
00201 {
00202     #if DEBUG_EXPERIMENT
00203         fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00204     #endif
00205     flags <= 1;
00206     if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00207     {
00208         if (ninputs && ninputs <= i)
00209         {
00210             experiment_error (experiment, _("bad templates number"));
00211             goto exit_on_error;
00212         }
00213         experiment->stencil[i]
00214             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00215         #if DEBUG_EXPERIMENT
00216             fprintf (stderr,
00217                 "experiment_open_xml: experiment=%s stencil%u=%s\n",
00218                 experiment->nexperiments, experiment->name,
00219                 experiment->stencil[i]);
00220         #endif
00221         ++experiment->ninputs;
00222         experiment->template_flags |= flags;
00223     }
00224     else if (xmlHasProp (node, (const xmlChar *) stencilbin[i]))
00225     {
00226         if (ninputs && ninputs <= i)
00227         {
00228             experiment_error (experiment, _("bad templates number"));
00229             goto exit_on_error;
00230         }
00231         experiment->stencil[i]
00232             = (char *) xmlGetProp (node, (const xmlChar *) stencilbin[i]);
00233         #if DEBUG_EXPERIMENT
00234             fprintf (stderr,
00235                 "experiment_open_xml: experiment=%s stencil%u=%s\n",
00236                 experiment->nexperiments, experiment->name,
00237                 experiment->stencil[i]);
00238         #endif
00239         ++experiment->ninputs;
00240     }
00241     else if (ninputs && ninputs > i)
00242     {
00243         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00244         experiment_error (experiment, buffer);
00245         goto exit_on_error;
00246     }
00247     else
00248         break;
00249 }
00250
00251 #if DEBUG_EXPERIMENT
00252     fprintf (stderr, "experiment_open_xml: end\n");
00253 #endif
00254     return 1;
00255
00256 exit_on_error:
00257     experiment_free (experiment, INPUT_TYPE_XML);
00258     #if DEBUG_EXPERIMENT
00259         fprintf (stderr, "experiment_open_xml: end\n");
00260     #endif
00261     return 0;
00262 }
00263
00269 int
00270 experiment_open_json (Experiment * experiment,
00271                     JsonNode * node,
00272                     unsigned int ninputs)
00273 {
00274     char buffer[64];
00275     JsonObject *object;
00276     const char *name;
00277     int error_code;
00278     unsigned int i;
00279     unsigned int flags = 1;
00280
00281     #if DEBUG_EXPERIMENT
00282         fprintf (stderr, "experiment_open_json: start\n");
00283     #endif
00284
00285

```

```

00286 // Resetting experiment data
00287 experiment_new (experiment);
00288
00289 // Getting JSON object
00290 object = json_node_get_object (node);
00291
00292 // Reading the experimental data
00293 name = json_object_get_string_member (object, LABEL_NAME);
00294 if (!name)
00295 {
00296     experiment_error (experiment, _("no data file name"));
00297     goto exit_on_error;
00298 }
00299 experiment->name = g_strdup (name);
00300 #if DEBUG_EXPERIMENT
00301 fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00302 #endif
00303 experiment->weight
00304 = jb_json_object_get_float_with_default (object, LABEL_WEIGHT, &error_code,
00305                                           1.);
00306 if (!error_code)
00307 {
00308     experiment_error (experiment, _("bad weight"));
00309     goto exit_on_error;
00310 }
00311 #if DEBUG_EXPERIMENT
00312 fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00313 #endif
00314 name = json_object_get_string_member (object, stencil[0]);
00315 if (name)
00316 {
00317     #if DEBUG_EXPERIMENT
00318     fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00319             name, stencil[0]);
00319     #endif
00320     ++experiment->ninputs;
00321     experiment->template_flags |= flags;
00322 }
00323 else
00324 {
00325     name = json_object_get_string_member (object, stencilbin[0]);
00326     if (name)
00327     {
00328         #if DEBUG_EXPERIMENT
00329         fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00330                 name, stencilbin[0]);
00331         #endif
00332         ++experiment->ninputs;
00333     }
00334     else
00335     {
00336         experiment_error (experiment, _("no template"));
00337         goto exit_on_error;
00338     }
00339 }
00340 experiment->stencil[0] = g_strdup (name);
00341 for (i = 1; i < MAX_NINPUTS; ++i)
00342 {
00343     #if DEBUG_EXPERIMENT
00344     fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00345     #endif
00346     flags <= 1;
00347     if (json_object_get_member (object, stencil[i]))
00348     {
00349         if (ninputs && ninputs <= i)
00350         {
00351             experiment_error (experiment, _("bad templates number"));
00352             goto exit_on_error;
00353         }
00354         name = json_object_get_string_member (object, stencil[i]);
00355         #if DEBUG_EXPERIMENT
00356         fprintf (stderr,
00357                 "experiment_open_json: experiment=%s stencil%u=%s\n",
00358                 experiment->nexperiments, name, stencil[i]);
00359         #endif
00360         experiment->stencil[i] = g_strdup (name);
00361         ++experiment->ninputs;
00362         experiment->template_flags |= flags;
00363     }
00364     else if (json_object_get_member (object, stencilbin[i]))
00365     {
00366         if (ninputs && ninputs <= i)
00367         {
00368             experiment_error (experiment, _("bad templates number"));
00369             goto exit_on_error;
00370         }
00371         name = json_object_get_string_member (object, stencilbin[i]);

```

```

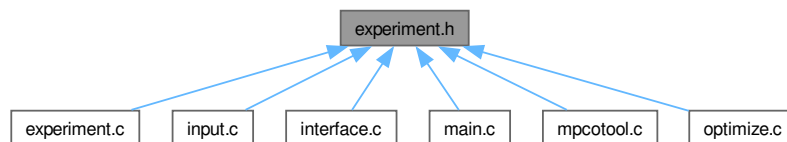
00373 #if DEBUG_EXPERIMENT
00374     fprintf (stderr,
00375             "experiment_open_json: experiment=%s stencil%u=%s\n",
00376             experiment->nexperiments, name, stencilbin[i]);
00377 #endif
00378     experiment->stencil[i] = g_strdup (name);
00379     ++experiment->ninputs;
00380 }
00381 else if (ninputs && ninputs > i)
00382 {
00383     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00384     experiment_error (experiment, buffer);
00385     goto exit_on_error;
00386 }
00387 else
00388     break;
00389 }
00390
00391 #if DEBUG_EXPERIMENT
00392     fprintf (stderr, "experiment_open_json: end\n");
00393 #endif
00394     return 1;
00395
00396 exit_on_error:
00397     experiment_free (experiment, INPUT_TYPE_JSON);
00398 #if DEBUG_EXPERIMENT
00399     fprintf (stderr, "experiment_open_json: end\n");
00400 #endif
00401     return 0;
00402 }

```

## 4.5 experiment.h File Reference

Header file to define the experiment data.

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



### Data Structures

- struct [Experiment](#)  
Struct to define the experiment data.

### Functions

- void [experiment\\_free](#) ([Experiment](#) \*experiment, unsigned int type)
- void [experiment\\_error](#) ([Experiment](#) \*experiment, char \*message)
- int [experiment\\_open\\_xml](#) ([Experiment](#) \*experiment, xmlNode \*node, unsigned int ninputs)
- int [experiment\\_open\\_json](#) ([Experiment](#) \*experiment, JsonNode \*node, unsigned int ninputs)

## Variables

- const char \* [stencil](#) [[MAX\\_NINPUTS](#)]  
*Array of strings with stencil labels.*
- const char \* [stencilbin](#) [[MAX\\_NINPUTS](#)]  
*Array of strings with binary stencil labels.*

## 4.5.1 Detailed Description

Header file to define the experiment data.

### Authors

Javier Burguete.

### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [experiment.h](#).

## 4.5.2 Function Documentation

### 4.5.2.1 [experiment\\_error\(\)](#)

```
void experiment_error (
    Experiment * experiment,
    char * message )
```

Function to print a message error opening an [Experiment](#) struct.

#### Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>message</i>	Error message.

Definition at line 117 of file [experiment.c](#).

```
00119 {
00120     if (!experiment->name)
00121         error_message = g_strconcat (_,("Experiment"), ":", message, NULL);
00122     else
00123         error_message = g_strconcat (_,("Experiment"), " ", experiment->name, ":",
00124                                     message, NULL);
00125 }
```

### 4.5.2.2 [experiment\\_free\(\)](#)

```
void experiment_free (
    Experiment * experiment,
    unsigned int type )
```



Function to free the memory of an [Experiment](#) struct.

## Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>type</i>	Type of input file.

Definition at line 88 of file [experiment.c](#).

```

00090 {
00091     unsigned int i;
00092     #if DEBUG_EXPERIMENT
00093         fprintf (stderr, "experiment_free: start\n");
00094     #endif
00095     if (type == INPUT_TYPE_XML)
00096     {
00097         for (i = 0; i < experiment->ninputs; ++i)
00098             xmlFree (experiment->stencil[i]);
00099         xmlFree (experiment->name);
00100     }
00101     else
00102     {
00103         for (i = 0; i < experiment->ninputs; ++i)
00104             g_free (experiment->stencil[i]);
00105         g_free (experiment->name);
00106     }
00107     experiment->ninputs = experiment->template_flags = 0;
00108     #if DEBUG_EXPERIMENT
00109         fprintf (stderr, "experiment_free: end\n");
00110     #endif
00111 }

```

#### 4.5.2.3 experiment\_open\_json()

```

int experiment_open_json (
    Experiment * experiment,
    JsonNode * node,
    unsigned int ninputs )

```

Function to open the [Experiment](#) struct on a XML node.

## Returns

1 on success, 0 on error.

## Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
<i>node</i>	JSON node.
<i>ninputs</i>	Number of the simulator input files.

Definition at line 270 of file [experiment.c](#).

```

00274 {
00275     char buffer[64];
00276     JsonObject *object;
00277     const char *name;
00278     int error_code;
00279     unsigned int i;
00280     unsigned int flags = 1;
00281
00282     #if DEBUG_EXPERIMENT
00283         fprintf (stderr, "experiment_open_json: start\n");
00284     #endif
00285
00286     // Resetting experiment data
00287     experiment\_new (experiment);
00288
00289     // Getting JSON object

```

```

00290     object = json_node_get_object (node);
00291
00292     // Reading the experimental data
00293     name = json_object_get_string_member (object, LABEL_NAME);
00294     if (!name)
00295     {
00296         experiment_error (experiment, _("no data file name"));
00297         goto exit_on_error;
00298     }
00299     experiment->name = g_strdup (name);
00300 #if DEBUG_EXPERIMENT
00301     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00302 #endif
00303     experiment->weight
00304     = jb_json_object_get_float_with_default (object, LABEL_WEIGHT, &error_code,
00305                                             1.);
00306     if (!error_code)
00307     {
00308         experiment_error (experiment, _("bad weight"));
00309         goto exit_on_error;
00310     }
00311 #if DEBUG_EXPERIMENT
00312     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00313 #endif
00314     name = json_object_get_string_member (object, stencil[0]);
00315     if (name)
00316     {
00317 #if DEBUG_EXPERIMENT
00318         fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00319                 name, stencil[0]);
00320 #endif
00321         ++experiment->ninputs;
00322         experiment->template_flags |= flags;
00323     }
00324     else
00325     {
00326         name = json_object_get_string_member (object, stencilbin[0]);
00327         if (name)
00328         {
00329 #if DEBUG_EXPERIMENT
00330             fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00331                     name, stencilbin[0]);
00332 #endif
00333             ++experiment->ninputs;
00334         }
00335         else
00336         {
00337             experiment_error (experiment, _("no template"));
00338             goto exit_on_error;
00339         }
00340     }
00341     experiment->stencil[0] = g_strdup (name);
00342     for (i = 1; i < MAX_NINPUTS; ++i)
00343     {
00344 #if DEBUG_EXPERIMENT
00345         fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00346 #endif
00347         flags <= 1;
00348         if (json_object_get_member (object, stencil[i]))
00349         {
00350             if (ninputs && ninputs <= i)
00351             {
00352                 experiment_error (experiment, _("bad templates number"));
00353                 goto exit_on_error;
00354             }
00355             name = json_object_get_string_member (object, stencil[i]);
00356 #if DEBUG_EXPERIMENT
00357             fprintf (stderr,
00358                     "experiment_open_json: experiment=%s stencil%u=%s\n",
00359                     experiment->nexperiments, name, stencil[i]);
00360 #endif
00361             experiment->stencil[i] = g_strdup (name);
00362             ++experiment->ninputs;
00363             experiment->template_flags |= flags;
00364         }
00365         else if (json_object_get_member (object, stencilbin[i]))
00366         {
00367             if (ninputs && ninputs <= i)
00368             {
00369                 experiment_error (experiment, _("bad templates number"));
00370                 goto exit_on_error;
00371             }
00372             name = json_object_get_string_member (object, stencilbin[i]);
00373 #if DEBUG_EXPERIMENT
00374             fprintf (stderr,
00375                     "experiment_open_json: experiment=%s stencil%u=%s\n",
00376                     experiment->nexperiments, name, stencilbin[i]);

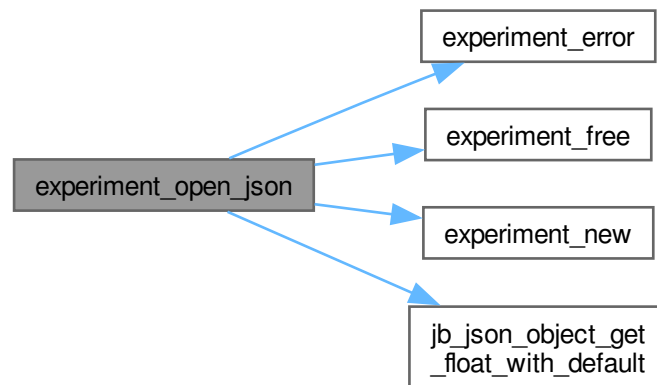
```

```

00377 #endif
00378     experiment->stencil[i] = g_strdup (name);
00379     ++experiment->ninputs;
00380 }
00381 else if (ninputs && ninputs > i)
00382 {
00383     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00384     experiment_error (experiment, buffer);
00385     goto exit_on_error;
00386 }
00387 else
00388     break;
00389 }
00390
00391 #if DEBUG_EXPERIMENT
00392 fprintf (stderr, "experiment_open_json: end\n");
00393 #endif
00394 return 1;
00395
00396 exit_on_error:
00397     experiment_free (experiment, INPUT_TYPE_JSON);
00398 #if DEBUG_EXPERIMENT
00399 fprintf (stderr, "experiment_open_json: end\n");
00400 #endif
00401 return 0;
00402 }

```

Here is the call graph for this function:



#### 4.5.2.4 experiment\_open\_xml()

```

int experiment_open_xml (
    Experiment * experiment,
    xmlNode * node,
    unsigned int ninputs )

```

Function to open the `Experiment` struct on a XML node.

##### Returns

1 on success, 0 on error.

## Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	XML node.
<i>ninputs</i>	Number of the simulator input files.

Definition at line 133 of file `experiment.c`.

```

00137 {
00138     char buffer[64];
00139     int error_code;
00140     unsigned int i;
00141     unsigned int flags = 1;
00142
00143     #if DEBUG_EXPERIMENT
00144         fprintf(stderr, "experiment_open_xml: start\n");
00145     #endif
00146
00147     // Resetting experiment data
00148     experiment_new (experiment);
00149
00150     // Reading the experimental data
00151     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00152     if (!experiment->name)
00153     {
00154         experiment_error (experiment, _("no data file name"));
00155         goto exit_on_error;
00156     }
00157     #if DEBUG_EXPERIMENT
00158         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00159     #endif
00160     experiment->weight
00161     = jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_WEIGHT,
00162                                         &error_code, 1.);
00163     if (!error_code)
00164     {
00165         experiment_error (experiment, _("bad weight"));
00166         goto exit_on_error;
00167     }
00168     #if DEBUG_EXPERIMENT
00169         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00170     #endif
00171     experiment->stencil[0]
00172     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00173     if (experiment->stencil[0])
00174     {
00175         #if DEBUG_EXPERIMENT
00176             fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00177                     experiment->name, stencil[0]);
00178         #endif
00179         ++experiment->ninputs;
00180         experiment->template_flags |= flags;
00181     }
00182     else
00183     {
00184         experiment->stencil[0]
00185         = (char *) xmlGetProp (node, (const xmlChar *) stencilbin[0]);
00186         if (experiment->stencil[0])
00187         {
00188             #if DEBUG_EXPERIMENT
00189                 fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00190                         experiment->name, stencilbin[0]);
00191             #endif
00192             ++experiment->ninputs;
00193         }
00194         else
00195         {
00196             experiment_error (experiment, _("no template"));
00197             goto exit_on_error;
00198         }
00199     }
00200     for (i = 1; i < MAX_NINPUTS; ++i)
00201     {
00202         #if DEBUG_EXPERIMENT
00203             fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00204         #endif
00205         flags <<= 1;
00206         if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00207         {
00208             if (ninputs && ninputs <= i)
00209             {
00210                 experiment_error (experiment, _("bad templates number"));
00211                 goto exit_on_error;

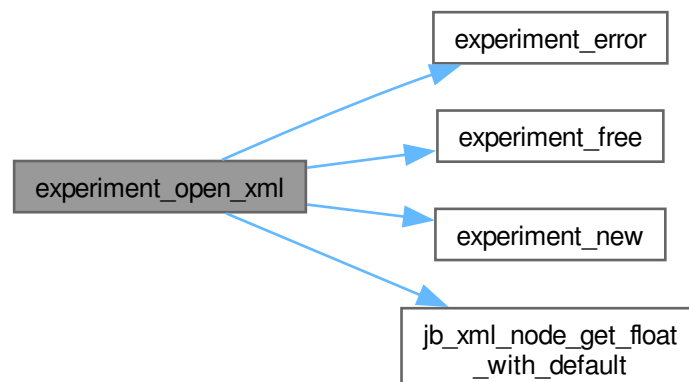
```

```

00212     }
00213     experiment->stencil[i]
00214     = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00215 #if DEBUG_EXPERIMENT
00216     fprintf (stderr,
00217             "experiment_open_xml: experiment=%s stencil%u=%s\n",
00218             experiment->nexperiments, experiment->name,
00219             experiment->stencil[i]);
00220 #endif
00221     ++experiment->ninputs;
00222     experiment->template_flags |= flags;
00223 }
00224 else if (xmlHasProp (node, (const xmlChar *) stencilbin[i]))
00225 {
00226     if (ninputs && ninputs <= i)
00227     {
00228         experiment_error (experiment, _("bad templates number"));
00229         goto exit_on_error;
00230     }
00231     experiment->stencil[i]
00232     = (char *) xmlGetProp (node, (const xmlChar *) stencilbin[i]);
00233 #if DEBUG_EXPERIMENT
00234     fprintf (stderr,
00235             "experiment_open_xml: experiment=%s stencil%u=%s\n",
00236             experiment->nexperiments, experiment->name,
00237             experiment->stencil[i]);
00238 #endif
00239     ++experiment->ninputs;
00240 }
00241 else if (ninputs && ninputs > i)
00242 {
00243     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00244     experiment_error (experiment, buffer);
00245     goto exit_on_error;
00246 }
00247 else
00248     break;
00249 }
00250
00251 #if DEBUG_EXPERIMENT
00252     fprintf (stderr, "experiment_open_xml: end\n");
00253 #endif
00254     return 1;
00255
00256 exit_on_error:
00257     experiment_free (experiment, INPUT_TYPE_XML);
00258 #if DEBUG_EXPERIMENT
00259     fprintf (stderr, "experiment_open_xml: end\n");
00260 #endif
00261     return 0;
00262 }

```

Here is the call graph for this function:



### 4.5.3 Variable Documentation

#### 4.5.3.1 stencil

```
const char* stencil[MAX_NINPUTS] [extern]
```

Array of strings with stencil labels.

Definition at line 54 of file [experiment.c](#).

```
00054 {
00055     LABEL_TEMPLATE1, LABEL_TEMPLATE2, LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00056     LABEL_TEMPLATE5, LABEL_TEMPLATE6, LABEL_TEMPLATE7, LABEL_TEMPLATE8
00057 };
```

#### 4.5.3.2 stencilbin

```
const char* stencilbin[MAX_NINPUTS] [extern]
```

Array of strings with binary stencil labels.

Definition at line 60 of file [experiment.c](#).

```
00060 {
00061     LABEL_INPUT1, LABEL_INPUT2, LABEL_INPUT3, LABEL_INPUT4,
00062     LABEL_INPUT5, LABEL_INPUT6, LABEL_INPUT7, LABEL_INPUT8
00063 };
```

## 4.6 experiment.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1
00034
00035 typedef struct
00036 {
```

```

00047 char *name;
00048 char *stencil[MAX_NINPUTS];
00049 double weight;
00050 unsigned int ninputs;
00051 unsigned int template_flags;
00052 } Experiment;
00053
00054 extern const char *stencil[MAX_NINPUTS];
00055 extern const char *stencilbin[MAX_NINPUTS];
00056
00057 // Public functions
00058 void experiment_free (Experiment * experiment, unsigned int type);
00059 void experiment_error (Experiment * experiment, char *message);
00060 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00061                        unsigned int ninputs);
00062 int experiment_open_json (Experiment * experiment, JsonNode * node,
00063                          unsigned int ninputs);
00064
00065 #endif

```

## 4.7 input.c File Reference

Source file to define the input functions.

```

#include "config.h"
#include <stdio.h>
#include <string.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include "jb/src/jb_xml.h"
#include "jb/src/jb_json.h"
#include "jb/src/jb_win.h"
#include "tools.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"

```

Include dependency graph for input.c:



### Macros

- `#define DEBUG_INPUT 0`  
Macro to debug input functions.

### Functions

- void `input_new` ()
- void `input_free` ()
- static void `input_error` (char \*message)
- static int `input_open_xml` (xmlDoc \*doc)
- static int `input_open_json` (JsonParser \*parser)
- int `input_open` (char \*filename)



## Variables

- [Input input](#) [1]  
*Global [Input](#) struct to set the input data.*
- `const char * result\_name = "result"`  
*Name of the result file.*
- `const char * variables\_name = "variables"`  
*Name of the variables file.*

## 4.7.1 Detailed Description

Source file to define the input functions.

### Authors

Javier Burguete and Borja Latorre.

### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [input.c](#).

## 4.7.2 Macro Definition Documentation

### 4.7.2.1 DEBUG\_INPUT

```
#define DEBUG_INPUT 0
```

Macro to debug input functions.

Definition at line [55](#) of file [input.c](#).

## 4.7.3 Function Documentation

### 4.7.3.1 input\_error()

```
static void input_error (  
    char * message ) [static]
```

Function to print an error message opening an [Input](#) struct.

### Parameters

<i>message</i>	Error message.
----------------	----------------

Definition at line 127 of file [input.c](#).

```
00128 {
00129     error_message = g_strconcat (_, "Input", ": ", message, "\n", NULL);
00130 }
```

#### 4.7.3.2 input\_free()

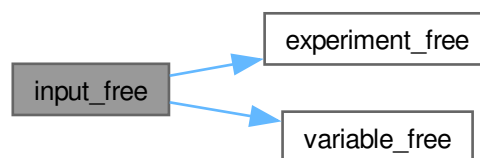
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 86 of file [input.c](#).

```
00087 {
00088     unsigned int i;
00089     #if DEBUG_INPUT
00090     fprintf (stderr, "input_free: start\n");
00091     #endif
00092     g_free (input->name);
00093     g_free (input->directory);
00094     for (i = 0; i < input->nexperiments; ++i)
00095         experiment_free (input->experiment + i, input->type);
00096     for (i = 0; i < input->nvariables; ++i)
00097         variable_free (input->variable + i, input->type);
00098     g_free (input->experiment);
00099     g_free (input->variable);
00100     if (input->type == INPUT_TYPE_XML)
00101     {
00102         xmlFree (input->cleaner);
00103         xmlFree (input->evaluator);
00104         xmlFree (input->simulator);
00105         xmlFree (input->result);
00106         xmlFree (input->variables);
00107     }
00108     else
00109     {
00110         g_free (input->cleaner);
00111         g_free (input->evaluator);
00112         g_free (input->simulator);
00113         g_free (input->result);
00114         g_free (input->variables);
00115     }
00116     input->nexperiments = input->nvariables = input->nsteps
00117         = input->nfinal_steps = 0;
00118     #if DEBUG_INPUT
00119     fprintf (stderr, "input_free: end\n");
00120     #endif
00121 }
```

Here is the call graph for this function:



### 4.7.3.3 input\_new()

```
void input_new ( )
```

Function to create a new [Input](#) struct.

Definition at line 66 of file [input.c](#).

```
00067 {
00068     #if DEBUG_INPUT
00069     fprintf (stderr, "input_new: start\n");
00070     #endif
00071     input->nvariables = input->nexperiments = input->nsteps = input->nfinal_steps
00072     = 0;
00073     input->simulator = input->evaluator = input->cleaner = input->directory
00074     = input->name = NULL;
00075     input->experiment = NULL;
00076     input->variable = NULL;
00077     #if DEBUG_INPUT
00078     fprintf (stderr, "input_new: end\n");
00079     #endif
00080 }
```

### 4.7.3.4 input\_open()

```
int input_open (
    char * filename )
```

Function to open the input file.

#### Returns

1\_on\_success, 0\_on\_error.

#### Parameters

<i>filename</i>	<a href="#">Input</a> data file name.
-----------------	---------------------------------------

Definition at line 1021 of file [input.c](#).

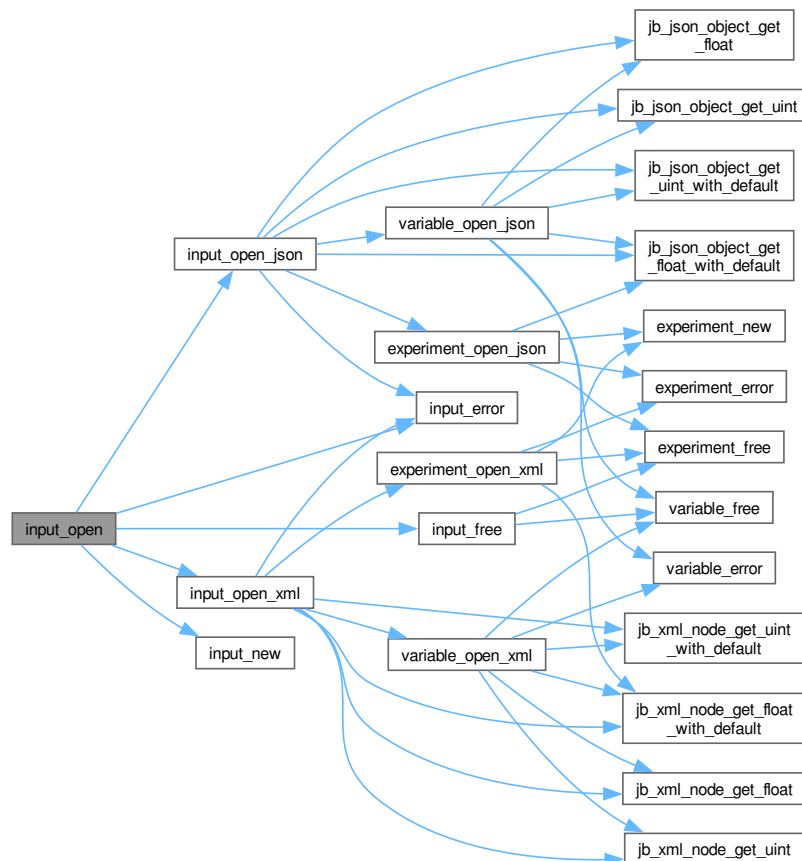
```
01022 {
01023     xmlDoc *doc;
01024     JsonParser *parser;
01025
01026     #if DEBUG_INPUT
01027     fprintf (stderr, "input_open: start\n");
01028     #endif
01029
01030     // Resetting input data
01031     input_new ();
01032
01033     // Opening input file
01034     #if DEBUG_INPUT
01035     fprintf (stderr, "input_open: opening the input file %s\n", filename);
01036     fprintf (stderr, "input_open: trying XML format\n");
01037     #endif
01038     doc = xmlParseFile (filename);
01039     if (!doc)
01040     {
01041         #if DEBUG_INPUT
01042         fprintf (stderr, "input_open: trying JSON format\n");
01043         #endif
01044         parser = json_parser_new ();
01045         if (!json_parser_load_from_file (parser, filename, NULL))
01046         {
01047             input_error (_("Unable to parse the input file"));
01048             goto exit_on_error;
01049         }
01050         if (!input_open_json (parser))
01051             goto exit_on_error;
```

```

01052     }
01053     else if (!input_open_xml (doc))
01054         goto exit_on_error;
01055
01056     // Getting the working directory
01057     input->directory = g_path_get_dirname (filename);
01058     input->name = g_path_get_basename (filename);
01059
01060 #if DEBUG_INPUT
01061     fprintf (stderr, "input_open: end\n");
01062 #endif
01063     return 1;
01064
01065 exit_on_error:
01066     jb_show_error (error_message);
01067     g_free (error_message);
01068     input_free ();
01069 #if DEBUG_INPUT
01070     fprintf (stderr, "input_open: end\n");
01071 #endif
01072     return 0;
01073 }

```

Here is the call graph for this function:



#### 4.7.3.5 input\_open\_json()

```

static int input_open_json (
    JsonParser * parser ) [inline], [static]

```

Function to open the input file in JSON format.

## Returns

1\_on\_success, 0\_on\_error.

## Parameters

<i>parser</i>	JsonParser struct.
---------------	--------------------

Definition at line 603 of file `input.c`.

```

00604 {
00605     Experiment *experiment;
00606     JsonNode *node, *child;
00607     JsonObject *object;
00608     JsonArray *array;
00609     const char *buffer;
00610     int error_code;
00611     unsigned int i, n;
00612
00613     #if DEBUG_INPUT
00614     fprintf (stderr, "input_open_json: start\n");
00615     #endif
00616
00617     // Resetting input data
00618     input->type = INPUT_TYPE_JSON;
00619
00620     // Getting the root node
00621     #if DEBUG_INPUT
00622     fprintf (stderr, "input_open_json: getting the root node\n");
00623     #endif
00624     node = json_parser_get_root (parser);
00625     object = json_node_get_object (node);
00626
00627     // Getting result and variables file names
00628     if (!input->result)
00629     {
00630         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00631         if (!buffer)
00632             buffer = result_name;
00633         input->result = g_strdup (buffer);
00634     }
00635     else
00636         input->result = g_strdup (result_name);
00637     if (!input->variables)
00638     {
00639         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00640         if (!buffer)
00641             buffer = variables_name;
00642         input->variables = g_strdup (buffer);
00643     }
00644     else
00645         input->variables = g_strdup (variables_name);
00646
00647     // Opening simulator program name
00648     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00649     if (!buffer)
00650     {
00651         input_error (_("Bad simulator program"));
00652         goto exit_on_error;
00653     }
00654     input->simulator = g_strdup (buffer);
00655
00656     // Opening evaluator program name
00657     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00658     if (buffer)
00659         input->evaluator = g_strdup (buffer);
00660
00661     // Opening cleaner program name
00662     buffer = json_object_get_string_member (object, LABEL_CLEANER);
00663     if (buffer)
00664         input->cleaner = g_strdup (buffer);
00665
00666     // Obtaining pseudo-random numbers generator seed
00667     input->seed
00668         = jb_json_object_get_uint_with_default (object, LABEL_SEED,
00669                                                &error_code, DEFAULT_RANDOM_SEED);
00670     if (!error_code)
00671     {
00672         input_error (_("Bad pseudo-random numbers generator seed"));
00673         goto exit_on_error;
00674     }
00675

```

```

00676 // Opening algorithm
00677 buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00678 if (!strcmp (buffer, LABEL_MONTE_CARLO))
00679 {
00680     input->algorithm = ALGORITHM_MONTE_CARLO;
00681
00682     // Obtaining simulations number
00683     input->nsimulations
00684     = jb_json_object_get_uint (object, LABEL_NSIMULATIONS, &error_code);
00685     if (!error_code)
00686     {
00687         input_error (_("Bad simulations number"));
00688         goto exit_on_error;
00689     }
00690 }
00691 else if (!strcmp (buffer, LABEL_SWEEP))
00692     input->algorithm = ALGORITHM_SWEEP;
00693 else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00694     input->algorithm = ALGORITHM_ORTHOGONAL;
00695 else if (!strcmp (buffer, LABEL_GENETIC))
00696 {
00697     input->algorithm = ALGORITHM_GENETIC;
00698
00699     // Obtaining population
00700     if (json_object_get_member (object, LABEL_NPOPULATION))
00701     {
00702         input->nsimulations
00703         = jb_json_object_get_uint (object, LABEL_NPOPULATION, &error_code);
00704         if (!error_code || input->nsimulations < 3)
00705         {
00706             input_error (_("Invalid population number"));
00707             goto exit_on_error;
00708         }
00709     }
00710     else
00711     {
00712         input_error (_("No population number"));
00713         goto exit_on_error;
00714     }
00715
00716     // Obtaining generations
00717     if (json_object_get_member (object, LABEL_NGENERATIONS))
00718     {
00719         input->niterations
00720         = jb_json_object_get_uint_with_default (object, LABEL_NGENERATIONS,
00721                                                 &error_code, 1);
00722         if (!error_code || !input->niterations)
00723         {
00724             input_error (_("Invalid generations number"));
00725             goto exit_on_error;
00726         }
00727     }
00728     else
00729     {
00730         input_error (_("No generations number"));
00731         goto exit_on_error;
00732     }
00733
00734     // Obtaining mutation probability
00735     if (json_object_get_member (object, LABEL_MUTATION))
00736     {
00737         input->mutation_ratio
00738         = jb_json_object_get_float (object, LABEL_MUTATION, &error_code);
00739         if (!error_code || input->mutation_ratio < 0.
00740             || input->mutation_ratio >= 1.)
00741         {
00742             input_error (_("Invalid mutation probability"));
00743             goto exit_on_error;
00744         }
00745     }
00746     else
00747     {
00748         input_error (_("No mutation probability"));
00749         goto exit_on_error;
00750     }
00751
00752     // Obtaining reproduction probability
00753     if (json_object_get_member (object, LABEL_REPRODUCTION))
00754     {
00755         input->reproduction_ratio
00756         = jb_json_object_get_float (object, LABEL_REPRODUCTION,
00757                                     &error_code);
00758         if (!error_code || input->reproduction_ratio < 0.
00759             || input->reproduction_ratio >= 1.0)
00760         {
00761             input_error (_("Invalid reproduction probability"));
00762             goto exit_on_error;

```

```

00763     }
00764 }
00765 else
00766 {
00767     input_error (_("No reproduction probability"));
00768     goto exit_on_error;
00769 }
00770
00771 // Obtaining adaptation probability
00772 if (json_object_get_member (object, LABEL_ADAPTATION))
00773 {
00774     input->adaptation_ratio
00775     = jb_json_object_get_float (object, LABEL_ADAPTATION, &error_code);
00776     if (!error_code || input->adaptation_ratio < 0.
00777         || input->adaptation_ratio >= 1.)
00778     {
00779         input_error (_("Invalid adaptation probability"));
00780         goto exit_on_error;
00781     }
00782 }
00783 else
00784 {
00785     input_error (_("No adaptation probability"));
00786     goto exit_on_error;
00787 }
00788
00789 // Checking survivals
00790 i = input->mutation_ratio * input->nsimulations;
00791 i += input->reproduction_ratio * input->nsimulations;
00792 i += input->adaptation_ratio * input->nsimulations;
00793 if (i > input->nsimulations - 2)
00794 {
00795     input_error
00796     (_("No enough survival entities to reproduce the population"));
00797     goto exit_on_error;
00798 }
00799 }
00800 else
00801 {
00802     input_error (_("Unknown algorithm"));
00803     goto exit_on_error;
00804 }
00805
00806 if (input->algorithm == ALGORITHM_MONTE_CARLO
00807     || input->algorithm == ALGORITHM_SWEEP
00808     || input->algorithm == ALGORITHM_ORTHOGONAL)
00809 {
00810
00811     // Obtaining iterations number
00812     input->niterations
00813     = jb_json_object_get_uint (object, LABEL_NITERATIONS, &error_code);
00814     if (!error_code || !input->niterations)
00815     {
00816         input_error (_("Bad iterations number"));
00817         goto exit_on_error;
00818     }
00819
00820     // Obtaining best number
00821     input->nbest
00822     = jb_json_object_get_uint_with_default (object, LABEL_NBEST,
00823                                             &error_code, 1);
00824     if (!error_code || !input->nbest)
00825     {
00826         input_error (_("Invalid best number"));
00827         goto exit_on_error;
00828     }
00829
00830     // Obtaining tolerance
00831     input->tolerance
00832     = jb_json_object_get_float_with_default (object, LABEL_TOLERANCE,
00833                                             &error_code, 0.);
00834     if (!error_code || input->tolerance < 0.)
00835     {
00836         input_error (_("Invalid tolerance"));
00837         goto exit_on_error;
00838     }
00839
00840     // Getting hill climbing method parameters
00841     if (json_object_get_member (object, LABEL_NSTEPS))
00842     {
00843         input->nsteps
00844         = jb_json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00845         if (!error_code)
00846         {
00847             input_error (_("Invalid steps number"));
00848             goto exit_on_error;
00849         }
00850     }

```

```

00850         if (json_object_has_member (object, LABEL_NFINAL_STEPS))
00851         {
00852             input->nfinal_steps
00853             = jb_json_object_get_uint (object, LABEL_NFINAL_STEPS,
00854                                     &error_code);
00855             if (!error_code)
00856             {
00857                 input_error (_("Invalid final steps number"));
00858                 goto exit_on_error;
00859             }
00860         }
00861         else
00862             input->nfinal_steps = input->nsteps;
00863         buffer = json_object_get_string_member (object, LABEL_CLIMBING);
00864         if (!strcmp (buffer, LABEL_COORDINATES))
00865             input->climbing = CLIMBING_METHOD_COORDINATES;
00866         else if (!strcmp (buffer, LABEL_RANDOM))
00867         {
00868             input->climbing = CLIMBING_METHOD_RANDOM;
00869             input->nestimates
00870             = jb_json_object_get_uint (object, LABEL_NESTIMATES,
00871                                     &error_code);
00872             if (!error_code || !input->nestimates)
00873             {
00874                 input_error (_("Invalid estimates number"));
00875                 goto exit_on_error;
00876             }
00877         }
00878         else
00879         {
00880             input_error (_("Unknown method to estimate the hill climbing"));
00881             goto exit_on_error;
00882         }
00883         input->relaxation
00884         = jb_json_object_get_float_with_default (object, LABEL_RELAXATION,
00885                                                 &error_code,
00886                                                 DEFAULT_RELAXATION);
00887         if (!error_code || input->relaxation < 0. || input->relaxation > 2.)
00888         {
00889             input_error (_("Invalid relaxation parameter"));
00890             goto exit_on_error;
00891         }
00892     }
00893     else
00894         input->nsteps = input->nfinal_steps = 0;
00895 }
00896 // Obtaining the threshold
00897 input->threshold
00898 = jb_json_object_get_float_with_default (object, LABEL_THRESHOLD,
00899                                         &error_code, 0.);
00900
00901 if (!error_code)
00902 {
00903     input_error (_("Invalid threshold"));
00904     goto exit_on_error;
00905 }
00906
00907 // Reading the experimental data
00908 array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00909 n = json_array_get_length (array);
00910 input->experiment = experiment = (Experiment *)
00911     g_malloc (n * sizeof (Experiment));
00912 for (i = 0; i < n; ++i)
00913 {
00914     #if DEBUG_INPUT
00915         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00916                 input->nexperiments);
00917     #endif
00918     child = json_array_get_element (array, i);
00919     if (!input->nexperiments)
00920     {
00921         if (!experiment_open_json (experiment, child, 0))
00922             goto exit_on_error;
00923     }
00924     else
00925     {
00926         if (!experiment_open_json (experiment + input->nexperiments,
00927                                 child, experiment->ninputs))
00928             goto exit_on_error;
00929         if (experiment[experiment->ninputs].template_flags
00930             != experiment->template_flags)
00931         {
00932             input_error ("bad template inputs");
00933             goto exit_on_error;
00934         }
00935     }
00936     ++input->nexperiments;

```

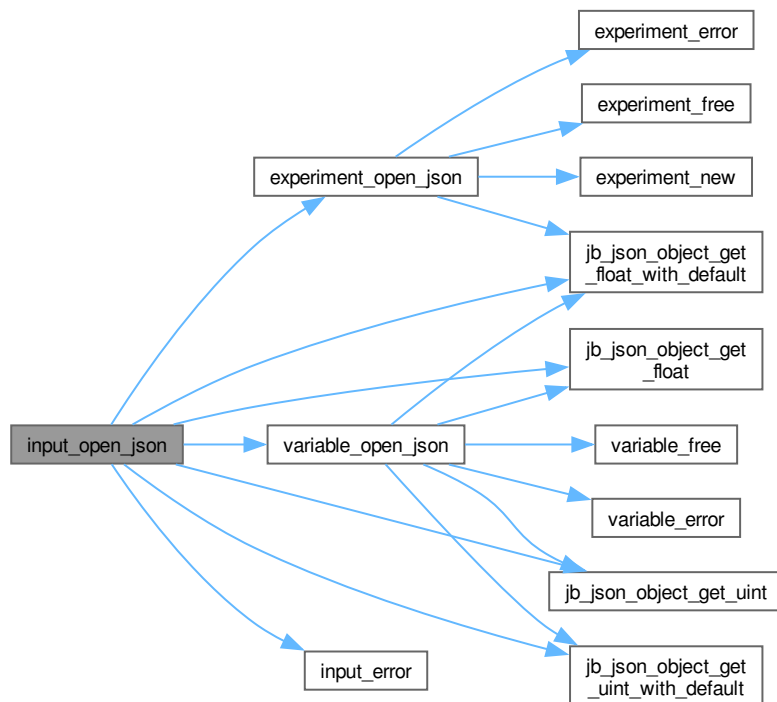


```

00937 #if DEBUG_INPUT
00938     fprintf (stderr, "input_open_json: nexperiments=%u\n",
00939             input->nexperiments);
00940 #endif
00941 }
00942 if (!input->nexperiments)
00943 {
00944     input_error (_("No optimization experiments"));
00945     goto exit_on_error;
00946 }
00947 input->template_flags = experiment->template_flags;
00948
00949 // Reading the variables data
00950 array = json_object_get_array_member (object, LABEL_VARIABLES);
00951 n = json_array_get_length (array);
00952 input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00953 for (i = 0; i < n; ++i)
00954 {
00955 #if DEBUG_INPUT
00956     fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00957 #endif
00958     child = json_array_get_element (array, i);
00959     if (!variable_open_json (input->variable + input->nvariables, child,
00960                             input->algorithm, input->nsteps))
00961         goto exit_on_error;
00962     ++input->nvariables;
00963 }
00964 if (!input->nvariables)
00965 {
00966     input_error (_("No optimization variables"));
00967     goto exit_on_error;
00968 }
00969
00970 // Obtaining the error norm
00971 if (json_object_get_member (object, LABEL_NORM))
00972 {
00973     buffer = json_object_get_string_member (object, LABEL_NORM);
00974     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00975         input->norm = ERROR_NORM_EUCLIDIAN;
00976     else if (!strcmp (buffer, LABEL_MAXIMUM))
00977         input->norm = ERROR_NORM_MAXIMUM;
00978     else if (!strcmp (buffer, LABEL_P))
00979     {
00980         input->norm = ERROR_NORM_P;
00981         input->p = jb_json_object_get_float (object, LABEL_P, &error_code);
00982         if (!error_code)
00983         {
00984             input_error (_("Bad P parameter"));
00985             goto exit_on_error;
00986         }
00987     }
00988     else if (!strcmp (buffer, LABEL_TAXICAB))
00989         input->norm = ERROR_NORM_TAXICAB;
00990     else
00991     {
00992         input_error (_("Unknown error norm"));
00993         goto exit_on_error;
00994     }
00995 }
00996 else
00997     input->norm = ERROR_NORM_EUCLIDIAN;
00998
00999 // Closing the JSON document
01000 g_object_unref (parser);
01001
01002 #if DEBUG_INPUT
01003     fprintf (stderr, "input_open_json: end\n");
01004 #endif
01005     return 1;
01006
01007 exit_on_error:
01008     g_object_unref (parser);
01009 #if DEBUG_INPUT
01010     fprintf (stderr, "input_open_json: end\n");
01011 #endif
01012     return 0;
01013 }

```

Here is the call graph for this function:



#### 4.7.3.6 input\_open\_xml()

```
static int input_open_xml (
    xmlDoc * doc ) [inline], [static]
```

Function to open the input file in XML format.

##### Returns

1\_on\_success, 0\_on\_error.

##### Parameters

<i>doc</i>	xmlDoc struct.
------------	----------------

Definition at line 138 of file `input.c`.

```
00139 {
00140     char buffer2[64];
00141     Experiment *experiment;
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i, nsteps;
00146
00147     #if DEBUG_INPUT
00148     fprintf (stderr, "input_open_xml: start\n");
```

```

00149 #endif
00150
00151 // Resetting input data
00152 buffer = NULL;
00153 input->type = INPUT_TYPE_XML;
00154
00155 // Getting the root node
00156 #if DEBUG_INPUT
00157 fprintf (stderr, "input_open_xml: getting the root node\n");
00158 #endif
00159 node = xmlDocGetRootElement (doc);
00160 if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161 {
00162     input_error (_("Bad root XML node"));
00163     goto exit_on_error;
00164 }
00165
00166 // Getting result and variables file names
00167 if (!input->result)
00168 {
00169     input->result =
00170         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171     if (!input->result)
00172         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173 }
00174 #if DEBUG_INPUT
00175 fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00176 #endif
00177 if (!input->variables)
00178 {
00179     input->variables =
00180         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00181     if (!input->variables)
00182         input->variables =
00183             (char *) xmlStrdup ((const xmlChar *) variables_name);
00184 }
00185 #if DEBUG_INPUT
00186 fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00187 #endif
00188
00189 // Opening simulator program name
00190 input->simulator =
00191     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00192 if (!input->simulator)
00193 {
00194     input_error (_("Bad simulator program"));
00195     goto exit_on_error;
00196 }
00197
00198 // Opening evaluator program name
00199 input->evaluator =
00200     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00201
00202 // Opening cleaner program name
00203 input->cleaner = (char *) xmlGetProp (node, (const xmlChar *) LABEL_CLEANER);
00204
00205 // Obtaining pseudo-random numbers generator seed
00206 input->seed
00207     = jb_xml_node_get_uint_with_default (node, (const xmlChar *) LABEL_SEED,
00208                                         &error_code, DEFAULT_RANDOM_SEED);
00209 if (!error_code)
00210 {
00211     input_error (_("Bad pseudo-random numbers generator seed"));
00212     goto exit_on_error;
00213 }
00214
00215 // Opening algorithm
00216 buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00217 if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00218 {
00219     input->algorithm = ALGORITHM_MONTE_CARLO;
00220 }
00221 // Obtaining simulations number
00222 input->nsimulations
00223     = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSIMULATIONS,
00224                             &error_code);
00225 if (!error_code)
00226 {
00227     input_error (_("Bad simulations number"));
00228     goto exit_on_error;
00229 }
00230 }
00231 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00232     input->algorithm = ALGORITHM_SWEEP;
00233 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00234     input->algorithm = ALGORITHM_ORTHOGONAL;
00235 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))

```

```

00236     {
00237         input->algorithm = ALGORITHM_GENETIC;
00238
00239         // Obtaining population
00240         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00241         {
00242             input->nsimulations
00243                 = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00244                                         &error_code);
00245             if (!error_code || input->nsimulations < 3)
00246             {
00247                 input_error (_("Invalid population number"));
00248                 goto exit_on_error;
00249             }
00250         }
00251         else
00252         {
00253             input_error (_("No population number"));
00254             goto exit_on_error;
00255         }
00256
00257         // Obtaining generations
00258         if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00259         {
00260             input->niterations
00261                 = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00262                                         &error_code);
00263             if (!error_code || !input->niterations)
00264             {
00265                 input_error (_("Invalid generations number"));
00266                 goto exit_on_error;
00267             }
00268         }
00269         else
00270         {
00271             input_error (_("No generations number"));
00272             goto exit_on_error;
00273         }
00274
00275         // Obtaining mutation probability
00276         if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00277         {
00278             input->mutation_ratio
00279                 = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00280                                         &error_code);
00281             if (!error_code || input->mutation_ratio < 0.
00282                 || input->mutation_ratio >= 1.)
00283             {
00284                 input_error (_("Invalid mutation probability"));
00285                 goto exit_on_error;
00286             }
00287         }
00288         else
00289         {
00290             input_error (_("No mutation probability"));
00291             goto exit_on_error;
00292         }
00293
00294         // Obtaining reproduction probability
00295         if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00296         {
00297             input->reproduction_ratio
00298                 = jb_xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00299                                         &error_code);
00300             if (!error_code || input->reproduction_ratio < 0.
00301                 || input->reproduction_ratio >= 1.0)
00302             {
00303                 input_error (_("Invalid reproduction probability"));
00304                 goto exit_on_error;
00305             }
00306         }
00307         else
00308         {
00309             input_error (_("No reproduction probability"));
00310             goto exit_on_error;
00311         }
00312
00313         // Obtaining adaptation probability
00314         if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00315         {
00316             input->adaptation_ratio
00317                 = jb_xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00318                                         &error_code);
00319             if (!error_code || input->adaptation_ratio < 0.
00320                 || input->adaptation_ratio >= 1.)
00321             {
00322                 input_error (_("Invalid adaptation probability"));

```

```

00323         goto exit_on_error;
00324     }
00325 }
00326 else
00327 {
00328     input_error (_("No adaptation probability"));
00329     goto exit_on_error;
00330 }
00331
00332 // Checking survivals
00333 i = input->mutation_ratio * input->nsimulations;
00334 i += input->reproduction_ratio * input->nsimulations;
00335 i += input->adaptation_ratio * input->nsimulations;
00336 if (i > input->nsimulations - 2)
00337 {
00338     input_error
00339     (_("No enough survival entities to reproduce the population"));
00340     goto exit_on_error;
00341 }
00342 }
00343 else
00344 {
00345     input_error (_("Unknown algorithm"));
00346     goto exit_on_error;
00347 }
00348 xmlFree (buffer);
00349 buffer = NULL;
00350
00351 if (input->algorithm == ALGORITHM_MONTE_CARLO
00352     || input->algorithm == ALGORITHM_SWEEP
00353     || input->algorithm == ALGORITHM_ORTHOGONAL)
00354 {
00355
00356     // Obtaining iterations number
00357     input->niterations = jb_xml_node_get_uint_with_default
00358     (node, (const xmlChar *) LABEL_NITERATIONS, &error_code, 1);
00359     if (!error_code || !input->niterations)
00360     {
00361         input_error (_("Bad iterations number"));
00362         goto exit_on_error;
00363     }
00364
00365     // Obtaining best number
00366     input->nbest
00367     = jb_xml_node_get_uint_with_default (node,
00368                                         (const xmlChar *) LABEL_NBEST,
00369                                         &error_code, 1);
00370     if (!error_code || !input->nbest)
00371     {
00372         input_error (_("Invalid best number"));
00373         goto exit_on_error;
00374     }
00375
00376     // Obtaining tolerance
00377     input->tolerance
00378     = jb_xml_node_get_float_with_default (node,
00379                                         (const xmlChar *) LABEL_TOLERANCE,
00380                                         &error_code, 0.);
00381     if (!error_code || input->tolerance < 0.)
00382     {
00383         input_error (_("Invalid tolerance"));
00384         goto exit_on_error;
00385     }
00386
00387     // Getting hill climbing method parameters
00388     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00389     {
00390         input->nsteps =
00391         jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00392                               &error_code);
00393         if (!error_code)
00394         {
00395             input_error (_("Invalid steps number"));
00396             goto exit_on_error;
00397         }
00398     }
00399     else
00400         input->nsteps = 0;
00401     if (xmlHasProp (node, (const xmlChar *) LABEL_NFINAL_STEPS))
00402     {
00403         input->nfinal_steps =
00404         jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NFINAL_STEPS,
00405                               &error_code);
00406         if (!error_code)
00407         {
00408             input_error (_("Invalid final steps number"));
00409             goto exit_on_error;

```

```

00410     }
00411 }
00412 else
00413     input->nfinal_steps = input->nsteps;
00414     nsteps = JBM_MAX (input->nsteps, input->nfinal_steps);
00415 #if DEBUG_INPUT
00416     fprintf (stderr, "input_open_xml: nsteps=%u\n", nsteps);
00417 #endif
00418     if (nsteps)
00419     {
00420         buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00421         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00422             input->climbing = CLIMBING_METHOD_COORDINATES;
00423         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00424         {
00425             input->climbing = CLIMBING_METHOD_RANDOM;
00426             input->nestimates
00427                 = jbm_xml_node_get_uint (node,
00428                                         (const xmlChar *) LABEL_NESTIMATES,
00429                                         &error_code);
00430             if (!error_code || !input->nestimates)
00431             {
00432                 input_error (_("Invalid estimates number"));
00433                 goto exit_on_error;
00434             }
00435         }
00436         else
00437         {
00438             input_error (_("Unknown method to estimate the hill climbing"));
00439             goto exit_on_error;
00440         }
00441         xmlFree (buffer);
00442         buffer = NULL;
00443         input->relaxation
00444             = jbm_xml_node_get_float_with_default (node,
00445                                                    (const xmlChar *)
00446                                                    LABEL_RELAXATION,
00447                                                    &error_code,
00448                                                    DEFAULT_RELAXATION);
00449         if (!error_code || input->relaxation < 0. || input->relaxation > 2.)
00450         {
00451             input_error (_("Invalid relaxation parameter"));
00452             goto exit_on_error;
00453         }
00454     }
00455 }
00456 // Obtaining the threshold
00457 input->threshold =
00458     jbm_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_THRESHOLD,
00459                                         &error_code, 0.);
00460 if (!error_code)
00461 {
00462     input_error (_("Invalid threshold"));
00463     goto exit_on_error;
00464 }
00465 // Reading the experimental data
00466 for (child = node->children; child; child = child->next)
00467 {
00468     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00469         break;
00470 #if DEBUG_INPUT
00471     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00472             input->nexperiments);
00473 #endif
00474     input->experiment = experiment = (Experiment *)
00475         g_realloc (input->experiment,
00476                   (1 + input->nexperiments) * sizeof (Experiment));
00477     if (!input->nexperiments)
00478     {
00479         if (!experiment_open_xml (experiment, child, 0))
00480             goto exit_on_error;
00481     }
00482     else
00483     {
00484         if (!experiment_open_xml (experiment + input->nexperiments,
00485                                 child, experiment->ninputs))
00486             goto exit_on_error;
00487         if (experiment[experiment->ninputs].template_flags
00488             != experiment->template_flags)
00489         {
00490             input_error ("bad template inputs");
00491             goto exit_on_error;
00492         }
00493     }
00494 }
00495 ++input->nexperiments;
00496 #if DEBUG_INPUT

```

```

00497     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00498               input->nexperiments);
00499 #endif
00500 }
00501 if (!input->nexperiments)
00502 {
00503     input_error (_("No optimization experiments"));
00504     goto exit_on_error;
00505 }
00506 input->template_flags = experiment->template_flags;
00507 buffer = NULL;
00508
00509 // Reading the variables data
00510 if (input->algorithm == ALGORITHM_SWEEP
00511     || input->algorithm == ALGORITHM_ORTHOGONAL)
00512     input->nsimulations = 1;
00513 for (; child; child = child->next)
00514 {
00515 #if DEBUG_INPUT
00516     fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00517 #endif
00518     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00519     {
00520         snprintf (buffer2, 64, "%s %u: %s",
00521                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00522         input_error (buffer2);
00523         goto exit_on_error;
00524     }
00525     input->variable = (Variable *)
00526         g_realloc (input->variable,
00527                   (1 + input->nvariables) * sizeof (Variable));
00528     if (!variable_open_xml (input->variable + input->nvariables, child,
00529                             input->algorithm, input->nsteps))
00530         goto exit_on_error;
00531     if (input->algorithm == ALGORITHM_SWEEP
00532         || input->algorithm == ALGORITHM_ORTHOGONAL)
00533         input->nsimulations *= input->variable[input->nvariables].nsweeps;
00534     ++input->nvariables;
00535 }
00536 if (!input->nvariables)
00537 {
00538     input_error (_("No optimization variables"));
00539     goto exit_on_error;
00540 }
00541 if (input->nbest > input->nsimulations)
00542 {
00543     input_error (_("Best number higher than simulations number"));
00544     goto exit_on_error;
00545 }
00546 buffer = NULL;
00547
00548 // Obtaining the error norm
00549 if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00550 {
00551     buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00552     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00553         input->norm = ERROR_NORM_EUCLIDIAN;
00554     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00555         input->norm = ERROR_NORM_MAXIMUM;
00556     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00557     {
00558         input->norm = ERROR_NORM_P;
00559         input->p
00560             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_P,
00561                                     &error_code);
00562         if (!error_code)
00563         {
00564             input_error (_("Bad P parameter"));
00565             goto exit_on_error;
00566         }
00567     }
00568     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00569         input->norm = ERROR_NORM_TAXICAB;
00570     else
00571     {
00572         input_error (_("Unknown error norm"));
00573         goto exit_on_error;
00574     }
00575     xmlFree (buffer);
00576 }
00577 else
00578     input->norm = ERROR_NORM_EUCLIDIAN;
00579
00580 // Closing the XML document
00581 xmlFreeDoc (doc);
00582
00583 #if DEBUG_INPUT

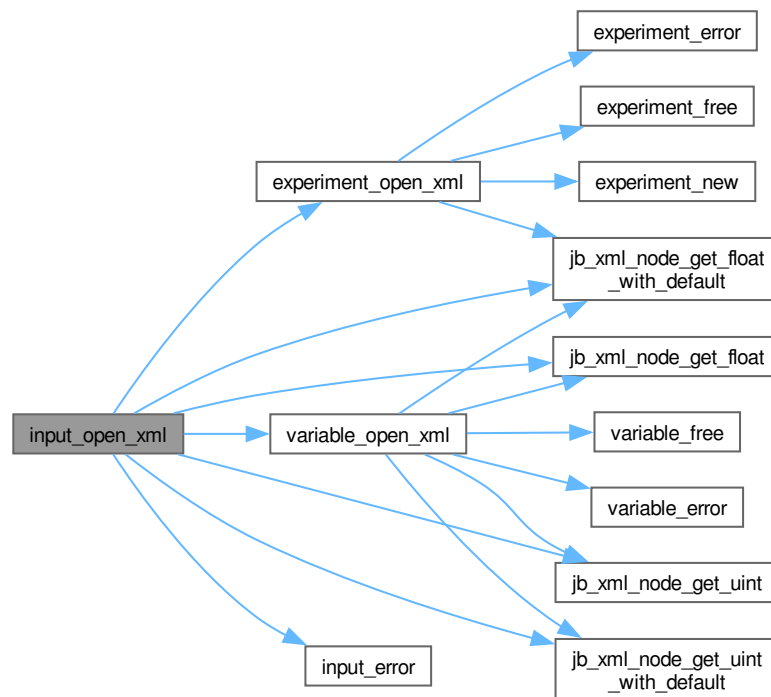
```

```

00584     fprintf (stderr, "input_open_xml: end\n");
00585 #endif
00586     return 1;
00587
00588 exit_on_error:
00589     xmlFree (buffer);
00590     xmlFreeDoc (doc);
00591 #if DEBUG_INPUT
00592     fprintf (stderr, "input_open_xml: end\n");
00593 #endif
00594     return 0;
00595 }

```

Here is the call graph for this function:



## 4.7.4 Variable Documentation

### 4.7.4.1 input

`Input` `input[1]`

Global `Input` struct to set the input data.

Definition at line 57 of file `input.c`.

### 4.7.4.2 result\_name

```
const char* result_name = "result"
```

Name of the result file.

Definition at line 59 of file `input.c`.



## 4.7.4.3 variables\_name

```
const char* variables_name = "variables"
```

Name of the variables file.

Definition at line 60 of file [input.c](#).

## 4.8 input.c

[Go to the documentation of this file.](#)

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <glib/gstdio.h>
00040 #include <json-glib/json-glib.h>
00041 #include "jb/src/jb_xml.h"
00042 #include "jb/src/jb_json.h"
00043 #include "jb/src/jb_win.h"
00044 #include "tools.h"
00045 #include "experiment.h"
00046 #include "variable.h"
00047 #include "input.h"
00048
00049 #define DEBUG_INPUT 0
00050
00051 Input input[1];
00052
00053 const char *result_name = "result";
00054 const char *variables_name = "variables";
00055
00056 void
00057 input_new ()
00058 {
00059     #if DEBUG_INPUT
00060         fprintf (stderr, "input_new: start\n");
00061     #endif
00062     input->nvariables = input->nexperiments = input->nsteps = input->nfinal_steps
00063         = 0;
00064     input->simulator = input->evaluator = input->cleaner = input->directory
00065         = input->name = NULL;
00066     input->experiment = NULL;
```

```

00076     input->variable = NULL;
00077     #if DEBUG_INPUT
00078     fprintf (stderr, "input_new: end\n");
00079     #endif
00080 }
00081
00085 void
00086 input_free ()
00087 {
00088     unsigned int i;
00089     #if DEBUG_INPUT
00090     fprintf (stderr, "input_free: start\n");
00091     #endif
00092     g_free (input->name);
00093     g_free (input->directory);
00094     for (i = 0; i < input->nexperiments; ++i)
00095         experiment_free (input->experiment + i, input->type);
00096     for (i = 0; i < input->nvariables; ++i)
00097         variable_free (input->variable + i, input->type);
00098     g_free (input->experiment);
00099     g_free (input->variable);
00100     if (input->type == INPUT_TYPE_XML)
00101     {
00102         xmlFree (input->cleaner);
00103         xmlFree (input->evaluator);
00104         xmlFree (input->simulator);
00105         xmlFree (input->result);
00106         xmlFree (input->variables);
00107     }
00108     else
00109     {
00110         g_free (input->cleaner);
00111         g_free (input->evaluator);
00112         g_free (input->simulator);
00113         g_free (input->result);
00114         g_free (input->variables);
00115     }
00116     input->nexperiments = input->nvariables = input->nsteps
00117     = input->nfinal_steps = 0;
00118     #if DEBUG_INPUT
00119     fprintf (stderr, "input_free: end\n");
00120     #endif
00121 }
00122
00126 static void
00127 input_error (char *message)
00128 {
00129     error_message = g_strconcat (_,("Input"), ":", message, "\n", NULL);
00130 }
00131
00137 static inline int
00138 input_open_xml (xmlDoc * doc)
00139 {
00140     char buffer2[64];
00141     Experiment *experiment;
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i, nsteps;
00146
00147     #if DEBUG_INPUT
00148     fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157     fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (_,("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173     }

```

```

00174 #if DEBUG_INPUT
00175     fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00176 #endif
00177     if (!input->variables)
00178     {
00179         input->variables =
00180             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00181         if (!input->variables)
00182             input->variables =
00183                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00184     }
00185 #if DEBUG_INPUT
00186     fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00187 #endif
00188
00189     // Opening simulator program name
00190     input->simulator =
00191         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00192     if (!input->simulator)
00193     {
00194         input_error (_("Bad simulator program"));
00195         goto exit_on_error;
00196     }
00197
00198     // Opening evaluator program name
00199     input->evaluator =
00200         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00201
00202     // Opening cleaner program name
00203     input->cleaner = (char *) xmlGetProp (node, (const xmlChar *) LABEL_CLEANER);
00204
00205     // Obtaining pseudo-random numbers generator seed
00206     input->seed
00207         = jb_xml_node_get_uint_with_default (node, (const xmlChar *) LABEL_SEED,
00208                                             &error_code, DEFAULT_RANDOM_SEED);
00209     if (!error_code)
00210     {
00211         input_error (_("Bad pseudo-random numbers generator seed"));
00212         goto exit_on_error;
00213     }
00214
00215     // Opening algorithm
00216     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00217     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00218     {
00219         input->algorithm = ALGORITHM_MONTE_CARLO;
00220
00221         // Obtaining simulations number
00222         input->nsimulations
00223             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSIMULATIONS,
00224                                     &error_code);
00225         if (!error_code)
00226         {
00227             input_error (_("Bad simulations number"));
00228             goto exit_on_error;
00229         }
00230     }
00231     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00232         input->algorithm = ALGORITHM_SWEEP;
00233     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00234         input->algorithm = ALGORITHM_ORTHOGONAL;
00235     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00236     {
00237         input->algorithm = ALGORITHM_GENETIC;
00238
00239         // Obtaining population
00240         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00241         {
00242             input->nsimulations
00243                 = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00244                                         &error_code);
00245             if (!error_code || input->nsimulations < 3)
00246             {
00247                 input_error (_("Invalid population number"));
00248                 goto exit_on_error;
00249             }
00250         }
00251     }
00252     else
00253     {
00254         input_error (_("No population number"));
00255         goto exit_on_error;
00256     }
00257
00258     // Obtaining generations
00259     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00260     {
00261         input->niterations

```

```

00261         = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00262                                 &error_code);
00263     if (!error_code || !input->niterations)
00264     {
00265         input_error (_("Invalid generations number"));
00266         goto exit_on_error;
00267     }
00268 }
00269 else
00270 {
00271     input_error (_("No generations number"));
00272     goto exit_on_error;
00273 }
00274
00275 // Obtaining mutation probability
00276 if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00277 {
00278     input->mutation_ratio
00279     = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00280                             &error_code);
00281     if (!error_code || input->mutation_ratio < 0.
00282         || input->mutation_ratio >= 1.)
00283     {
00284         input_error (_("Invalid mutation probability"));
00285         goto exit_on_error;
00286     }
00287 }
00288 else
00289 {
00290     input_error (_("No mutation probability"));
00291     goto exit_on_error;
00292 }
00293
00294 // Obtaining reproduction probability
00295 if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00296 {
00297     input->reproduction_ratio
00298     = jb_xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00299                             &error_code);
00300     if (!error_code || input->reproduction_ratio < 0.
00301         || input->reproduction_ratio >= 1.0)
00302     {
00303         input_error (_("Invalid reproduction probability"));
00304         goto exit_on_error;
00305     }
00306 }
00307 else
00308 {
00309     input_error (_("No reproduction probability"));
00310     goto exit_on_error;
00311 }
00312
00313 // Obtaining adaptation probability
00314 if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00315 {
00316     input->adaptation_ratio
00317     = jb_xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00318                             &error_code);
00319     if (!error_code || input->adaptation_ratio < 0.
00320         || input->adaptation_ratio >= 1.)
00321     {
00322         input_error (_("Invalid adaptation probability"));
00323         goto exit_on_error;
00324     }
00325 }
00326 else
00327 {
00328     input_error (_("No adaptation probability"));
00329     goto exit_on_error;
00330 }
00331
00332 // Checking survivals
00333 i = input->mutation_ratio * input->nsimulations;
00334 i += input->reproduction_ratio * input->nsimulations;
00335 i += input->adaptation_ratio * input->nsimulations;
00336 if (i > input->nsimulations - 2)
00337 {
00338     input_error
00339     (_("No enough survival entities to reproduce the population"));
00340     goto exit_on_error;
00341 }
00342 }
00343 else
00344 {
00345     input_error (_("Unknown algorithm"));
00346     goto exit_on_error;
00347 }

```

```

00348     xmlFree (buffer);
00349     buffer = NULL;
00350
00351     if (input->algorithm == ALGORITHM_MONTE_CARLO
00352         || input->algorithm == ALGORITHM_SWEEP
00353         || input->algorithm == ALGORITHM_ORTHOGONAL)
00354     {
00355
00356         // Obtaining iterations number
00357         input->niterations = jb_xml_node_get_uint_with_default
00358             (node, (const xmlChar *) LABEL_NITERATIONS, &error_code, 1);
00359         if (!error_code || !input->niterations)
00360         {
00361             input_error (_("Bad iterations number"));
00362             goto exit_on_error;
00363         }
00364
00365         // Obtaining best number
00366         input->nbest
00367             = jb_xml_node_get_uint_with_default (node,
00368                                                 (const xmlChar *) LABEL_NBEST,
00369                                                 &error_code, 1);
00370         if (!error_code || !input->nbest)
00371         {
00372             input_error (_("Invalid best number"));
00373             goto exit_on_error;
00374         }
00375
00376         // Obtaining tolerance
00377         input->tolerance
00378             = jb_xml_node_get_float_with_default (node,
00379                                                 (const xmlChar *) LABEL_TOLERANCE,
00380                                                 &error_code, 0.);
00381         if (!error_code || input->tolerance < 0.)
00382         {
00383             input_error (_("Invalid tolerance"));
00384             goto exit_on_error;
00385         }
00386
00387         // Getting hill climbing method parameters
00388         if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00389         {
00390             input->nsteps =
00391                 jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00392                                     &error_code);
00393             if (!error_code)
00394             {
00395                 input_error (_("Invalid steps number"));
00396                 goto exit_on_error;
00397             }
00398         }
00399         else
00400             input->nsteps = 0;
00401         if (xmlHasProp (node, (const xmlChar *) LABEL_NFINAL_STEPS))
00402         {
00403             input->nfinal_steps =
00404                 jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NFINAL_STEPS,
00405                                     &error_code);
00406             if (!error_code)
00407             {
00408                 input_error (_("Invalid final steps number"));
00409                 goto exit_on_error;
00410             }
00411         }
00412         else
00413             input->nfinal_steps = input->nsteps;
00414         nsteps = JBM_MAX (input->nsteps, input->nfinal_steps);
00415         #if DEBUG_INPUT
00416         fprintf (stderr, "input_open_xml: nsteps=%u\n", nsteps);
00417         #endif
00418         if (nsteps)
00419         {
00420             buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00421             if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00422                 input->climbing = CLIMBING_METHOD_COORDINATES;
00423             else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00424             {
00425                 input->climbing = CLIMBING_METHOD_RANDOM;
00426                 input->nestimates
00427                     = jb_xml_node_get_uint (node,
00428                                             (const xmlChar *) LABEL_NESTIMATES,
00429                                             &error_code);
00430                 if (!error_code || !input->nestimates)
00431                 {
00432                     input_error (_("Invalid estimates number"));
00433                     goto exit_on_error;
00434                 }

```

```

00435         }
00436     else
00437     {
00438         input_error (_("Unknown method to estimate the hill climbing"));
00439         goto exit_on_error;
00440     }
00441     xmlFree (buffer);
00442     buffer = NULL;
00443     input->relaxation
00444     = jb_xml_node_get_float_with_default (node,
00445         (const xmlChar *)
00446         LABEL_RELAXATION,
00447         &error_code,
00448         DEFAULT_RELAXATION);
00449     if (!error_code || input->relaxation < 0. || input->relaxation > 2.)
00450     {
00451         input_error (_("Invalid relaxation parameter"));
00452         goto exit_on_error;
00453     }
00454 }
00455 }
00456 // Obtaining the threshold
00457 input->threshold =
00458     jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_THRESHOLD,
00459         &error_code, 0.);
00460 if (!error_code)
00461 {
00462     input_error (_("Invalid threshold"));
00463     goto exit_on_error;
00464 }
00465 // Reading the experimental data
00466 for (child = node->children; child; child = child->next)
00467 {
00468     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00469         break;
00470 #if DEBUG_INPUT
00471     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00472         input->nexperiments);
00473 #endif
00474     input->experiment = experiment = (Experiment *)
00475         g_realloc (input->experiment,
00476             (1 + input->nexperiments) * sizeof (Experiment));
00477     if (!input->nexperiments)
00478     {
00479         if (!experiment_open_xml (experiment, child, 0))
00480             goto exit_on_error;
00481     }
00482     else
00483     {
00484         if (!experiment_open_xml (experiment + input->nexperiments,
00485             child, experiment->ninputs))
00486             goto exit_on_error;
00487         if (experiment[experiment->ninputs].template_flags
00488             != experiment->template_flags)
00489         {
00490             input_error ("bad template inputs");
00491             goto exit_on_error;
00492         }
00493     }
00494     ++input->nexperiments;
00495 #if DEBUG_INPUT
00496     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00497         input->nexperiments);
00498 #endif
00499 }
00500 if (!input->nexperiments)
00501 {
00502     input_error (_("No optimization experiments"));
00503     goto exit_on_error;
00504 }
00505 input->template_flags = experiment->template_flags;
00506 buffer = NULL;
00507 // Reading the variables data
00508 if (input->algorithm == ALGORITHM_SWEEP
00509     || input->algorithm == ALGORITHM_ORTHOGONAL)
00510     input->nsimulations = 1;
00511 for (; child; child = child->next)
00512 {
00513     #if DEBUG_INPUT
00514         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00515     #endif
00516     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00517     {
00518         snprintf (buffer2, 64, "%s %u: %s",
00519             _("Variable"), input->nvariables + 1, _("bad XML node"));
00520     }

```

```

00522         input_error (buffer2);
00523         goto exit_on_error;
00524     }
00525     input->variable = (Variable *)
00526         g_realloc (input->variable,
00527             (1 + input->nvariables) * sizeof (Variable));
00528     if (!variable_open_xml (input->variable + input->nvariables, child,
00529         input->algorithm, input->nsteps))
00530         goto exit_on_error;
00531     if (input->algorithm == ALGORITHM_SWEEP
00532         || input->algorithm == ALGORITHM_ORTHOGONAL)
00533         input->nsimulations += input->variable[input->nvariables].nsweeps;
00534     ++input->nvariables;
00535 }
00536 if (!input->nvariables)
00537 {
00538     input_error (_("No optimization variables"));
00539     goto exit_on_error;
00540 }
00541 if (input->nbest > input->nsimulations)
00542 {
00543     input_error (_("Best number higher than simulations number"));
00544     goto exit_on_error;
00545 }
00546 buffer = NULL;
00547 // Obtaining the error norm
00548 if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00549 {
00550     buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00551     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00552         input->norm = ERROR_NORM_EUCLIDIAN;
00553     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00554         input->norm = ERROR_NORM_MAXIMUM;
00555     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00556     {
00557         input->norm = ERROR_NORM_P;
00558         input->p
00559             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_P,
00560                 &error_code);
00561         if (!error_code)
00562         {
00563             input_error (_("Bad P parameter"));
00564             goto exit_on_error;
00565         }
00566     }
00567     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00568         input->norm = ERROR_NORM_TAXICAB;
00569     else
00570     {
00571         input_error (_("Unknown error norm"));
00572         goto exit_on_error;
00573     }
00574     xmlFree (buffer);
00575 }
00576 else
00577     input->norm = ERROR_NORM_EUCLIDIAN;
00578 // Closing the XML document
00579 xmlFreeDoc (doc);
00580 #if DEBUG_INPUT
00581     fprintf (stderr, "input_open_xml: end\n");
00582 #endif
00583     return 1;
00584 }
00585 exit_on_error:
00586     xmlFree (buffer);
00587     xmlFreeDoc (doc);
00588 #if DEBUG_INPUT
00589     fprintf (stderr, "input_open_xml: end\n");
00590 #endif
00591     return 0;
00592 }
00593
00594 static inline int
00595 input_open_json (JsonParser * parser)
00596 {
00597     Experiment *experiment;
00598     JsonNode *node, *child;
00599     JsonObject *object;
00600     JsonArray *array;
00601     const char *buffer;
00602     int error_code;
00603     unsigned int i, n;
00604 #if DEBUG_INPUT

```

```

00614     fprintf (stderr, "input_open_json: start\n");
00615 #endif
00616
00617     // Resetting input data
00618     input->type = INPUT_TYPE_JSON;
00619
00620     // Getting the root node
00621 #if DEBUG_INPUT
00622     fprintf (stderr, "input_open_json: getting the root node\n");
00623 #endif
00624     node = json_parser_get_root (parser);
00625     object = json_node_get_object (node);
00626
00627     // Getting result and variables file names
00628     if (!input->result)
00629     {
00630         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00631         if (!buffer)
00632             buffer = result_name;
00633         input->result = g_strdup (buffer);
00634     }
00635     else
00636         input->result = g_strdup (result_name);
00637     if (!input->variables)
00638     {
00639         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00640         if (!buffer)
00641             buffer = variables_name;
00642         input->variables = g_strdup (buffer);
00643     }
00644     else
00645         input->variables = g_strdup (variables_name);
00646
00647     // Opening simulator program name
00648     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00649     if (!buffer)
00650     {
00651         input_error (_("Bad simulator program"));
00652         goto exit_on_error;
00653     }
00654     input->simulator = g_strdup (buffer);
00655
00656     // Opening evaluator program name
00657     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00658     if (buffer)
00659         input->evaluator = g_strdup (buffer);
00660
00661     // Opening cleaner program name
00662     buffer = json_object_get_string_member (object, LABEL_CLEANER);
00663     if (buffer)
00664         input->cleaner = g_strdup (buffer);
00665
00666     // Obtaining pseudo-random numbers generator seed
00667     input->seed
00668     = jb_json_object_get_uint_with_default (object, LABEL_SEED,
00669                                             &error_code, DEFAULT_RANDOM_SEED);
00670     if (!error_code)
00671     {
00672         input_error (_("Bad pseudo-random numbers generator seed"));
00673         goto exit_on_error;
00674     }
00675
00676     // Opening algorithm
00677     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00678     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00679     {
00680         input->algorithm = ALGORITHM_MONTE_CARLO;
00681
00682         // Obtaining simulations number
00683         input->nsimulations
00684         = jb_json_object_get_uint (object, LABEL_NSIMULATIONS, &error_code);
00685         if (!error_code)
00686         {
00687             input_error (_("Bad simulations number"));
00688             goto exit_on_error;
00689         }
00690     }
00691     else if (!strcmp (buffer, LABEL_SWEEP))
00692         input->algorithm = ALGORITHM_SWEEP;
00693     else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00694         input->algorithm = ALGORITHM_ORTHOGONAL;
00695     else if (!strcmp (buffer, LABEL_GENETIC))
00696     {
00697         input->algorithm = ALGORITHM_GENETIC;
00698
00699         // Obtaining population
00700         if (json_object_get_member (object, LABEL_NPOPULATION))

```



```

00701     {
00702         input->nsimulations
00703         = jb_json_object_get_uint (object, LABEL_NPOPULATION, &error_code);
00704         if (!error_code || input->nsimulations < 3)
00705         {
00706             input_error (_("Invalid population number"));
00707             goto exit_on_error;
00708         }
00709     }
00710     else
00711     {
00712         input_error (_("No population number"));
00713         goto exit_on_error;
00714     }
00715
00716     // Obtaining generations
00717     if (json_object_get_member (object, LABEL_NGENERATIONS))
00718     {
00719         input->niterations
00720         = jb_json_object_get_uint_with_default (object, LABEL_NGENERATIONS,
00721                                                 &error_code, 1);
00722         if (!error_code || !input->niterations)
00723         {
00724             input_error (_("Invalid generations number"));
00725             goto exit_on_error;
00726         }
00727     }
00728     else
00729     {
00730         input_error (_("No generations number"));
00731         goto exit_on_error;
00732     }
00733
00734     // Obtaining mutation probability
00735     if (json_object_get_member (object, LABEL_MUTATION))
00736     {
00737         input->mutation_ratio
00738         = jb_json_object_get_float (object, LABEL_MUTATION, &error_code);
00739         if (!error_code || input->mutation_ratio < 0.
00740             || input->mutation_ratio >= 1.)
00741         {
00742             input_error (_("Invalid mutation probability"));
00743             goto exit_on_error;
00744         }
00745     }
00746     else
00747     {
00748         input_error (_("No mutation probability"));
00749         goto exit_on_error;
00750     }
00751
00752     // Obtaining reproduction probability
00753     if (json_object_get_member (object, LABEL_REPRODUCTION))
00754     {
00755         input->reproduction_ratio
00756         = jb_json_object_get_float (object, LABEL_REPRODUCTION,
00757                                     &error_code);
00758         if (!error_code || input->reproduction_ratio < 0.
00759             || input->reproduction_ratio >= 1.0)
00760         {
00761             input_error (_("Invalid reproduction probability"));
00762             goto exit_on_error;
00763         }
00764     }
00765     else
00766     {
00767         input_error (_("No reproduction probability"));
00768         goto exit_on_error;
00769     }
00770
00771     // Obtaining adaptation probability
00772     if (json_object_get_member (object, LABEL_ADAPTATION))
00773     {
00774         input->adaptation_ratio
00775         = jb_json_object_get_float (object, LABEL_ADAPTATION, &error_code);
00776         if (!error_code || input->adaptation_ratio < 0.
00777             || input->adaptation_ratio >= 1.)
00778         {
00779             input_error (_("Invalid adaptation probability"));
00780             goto exit_on_error;
00781         }
00782     }
00783     else
00784     {
00785         input_error (_("No adaptation probability"));
00786         goto exit_on_error;
00787     }

```

```

00788
00789 // Checking survivals
00790 i = input->mutation_ratio * input->nsimulations;
00791 i += input->reproduction_ratio * input->nsimulations;
00792 i += input->adaptation_ratio * input->nsimulations;
00793 if (i > input->nsimulations - 2)
00794 {
00795     input_error
00796         (_("No enough survival entities to reproduce the population"));
00797     goto exit_on_error;
00798 }
00799 }
00800 else
00801 {
00802     input_error (_("Unknown algorithm"));
00803     goto exit_on_error;
00804 }
00805
00806 if (input->algorithm == ALGORITHM_MONTE_CARLO
00807     || input->algorithm == ALGORITHM_SWEEP
00808     || input->algorithm == ALGORITHM_ORTHOGONAL)
00809 {
00810
00811     // Obtaining iterations number
00812     input->niterations
00813         = jb_json_object_get_uint (object, LABEL_NITERATIONS, &error_code);
00814     if (!error_code || !input->niterations)
00815     {
00816         input_error (_("Bad iterations number"));
00817         goto exit_on_error;
00818     }
00819
00820     // Obtaining best number
00821     input->nbest
00822         = jb_json_object_get_uint_with_default (object, LABEL_NBEST,
00823         &error_code, 1);
00824     if (!error_code || !input->nbest)
00825     {
00826         input_error (_("Invalid best number"));
00827         goto exit_on_error;
00828     }
00829
00830     // Obtaining tolerance
00831     input->tolerance
00832         = jb_json_object_get_float_with_default (object, LABEL_TOLERANCE,
00833         &error_code, 0.);
00834     if (!error_code || input->tolerance < 0.)
00835     {
00836         input_error (_("Invalid tolerance"));
00837         goto exit_on_error;
00838     }
00839
00840     // Getting hill climbing method parameters
00841     if (json_object_get_member (object, LABEL_NSTEPS))
00842     {
00843         input->nsteps
00844             = jb_json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00845         if (!error_code)
00846         {
00847             input_error (_("Invalid steps number"));
00848             goto exit_on_error;
00849         }
00850         if (json_object_has_member (object, LABEL_NFINAL_STEPS))
00851         {
00852             input->nfinal_steps
00853                 = jb_json_object_get_uint (object, LABEL_NFINAL_STEPS,
00854                 &error_code);
00855             if (!error_code)
00856             {
00857                 input_error (_("Invalid final steps number"));
00858                 goto exit_on_error;
00859             }
00860         }
00861         else
00862             input->nfinal_steps = input->nsteps;
00863         buffer = json_object_get_string_member (object, LABEL_CLIMBING);
00864         if (!strcmp (buffer, LABEL_COORDINATES))
00865             input->climbing = CLIMBING_METHOD_COORDINATES;
00866         else if (!strcmp (buffer, LABEL_RANDOM))
00867         {
00868             input->climbing = CLIMBING_METHOD_RANDOM;
00869             input->nestimates
00870                 = jb_json_object_get_uint (object, LABEL_NESTIMATES,
00871                 &error_code);
00872             if (!error_code || !input->nestimates)
00873             {
00874                 input_error (_("Invalid estimates number"));

```

```

00875         goto exit_on_error;
00876     }
00877 }
00878 else
00879 {
00880     input_error (_, "Unknown method to estimate the hill climbing");
00881     goto exit_on_error;
00882 }
00883 input->relaxation
00884 = jb_json_object_get_float_with_default (object, LABEL_RELAXATION,
00885                                         &error_code,
00886                                         DEFAULT_RELAXATION);
00887 if (!error_code || input->relaxation < 0. || input->relaxation > 2.)
00888 {
00889     input_error (_, "Invalid relaxation parameter");
00890     goto exit_on_error;
00891 }
00892 }
00893 else
00894     input->nsteps = input->nfinal_steps = 0;
00895 }
00896 // Obtaining the threshold
00897 input->threshold
00898 = jb_json_object_get_float_with_default (object, LABEL_THRESHOLD,
00899                                         &error_code, 0.);
00900 if (!error_code)
00901 {
00902     input_error (_, "Invalid threshold");
00903     goto exit_on_error;
00904 }
00905 }
00906 // Reading the experimental data
00907 array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00908 n = json_array_get_length (array);
00909 input->experiment = experiment = (Experiment *)
00910     g_malloc (n * sizeof (Experiment));
00911 for (i = 0; i < n; ++i)
00912 {
00913     #if DEBUG_INPUT
00914     fprintf (stderr, "input_open_json: nexperiments=%u\n",
00915             input->nexperiments);
00916     #endif
00917     child = json_array_get_element (array, i);
00918     if (!input->nexperiments)
00919     {
00920         if (!experiment_open_json (experiment, child, 0))
00921             goto exit_on_error;
00922     }
00923     else
00924     {
00925         if (!experiment_open_json (experiment + input->nexperiments,
00926                                   child, experiment->ninputs))
00927             goto exit_on_error;
00928         if (experiment[experiment->ninputs].template_flags
00929             != experiment->template_flags)
00930         {
00931             input_error ("bad template inputs");
00932             goto exit_on_error;
00933         }
00934     }
00935     ++input->nexperiments;
00936     #if DEBUG_INPUT
00937     fprintf (stderr, "input_open_json: nexperiments=%u\n",
00938             input->nexperiments);
00939     #endif
00940 }
00941 if (!input->nexperiments)
00942 {
00943     input_error (_, "No optimization experiments");
00944     goto exit_on_error;
00945 }
00946 input->template_flags = experiment->template_flags;
00947 // Reading the variables data
00948 array = json_object_get_array_member (object, LABEL_VARIABLES);
00949 n = json_array_get_length (array);
00950 input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00951 for (i = 0; i < n; ++i)
00952 {
00953     #if DEBUG_INPUT
00954     fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00955     #endif
00956     child = json_array_get_element (array, i);
00957     if (!variable_open_json (input->variable + input->nvariables, child,
00958                             input->algorithm, input->nsteps))
00959         goto exit_on_error;
00960 }

```

```

00962     ++input->nvariables;
00963 }
00964 if (!input->nvariables)
00965 {
00966     input_error (_("No optimization variables"));
00967     goto exit_on_error;
00968 }
00969
00970 // Obtaining the error norm
00971 if (json_object_get_member (object, LABEL_NORM))
00972 {
00973     buffer = json_object_get_string_member (object, LABEL_NORM);
00974     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00975         input->norm = ERROR_NORM_EUCLIDIAN;
00976     else if (!strcmp (buffer, LABEL_MAXIMUM))
00977         input->norm = ERROR_NORM_MAXIMUM;
00978     else if (!strcmp (buffer, LABEL_P))
00979     {
00980         input->norm = ERROR_NORM_P;
00981         input->p = jb_json_object_get_float (object, LABEL_P, &error_code);
00982         if (!error_code)
00983         {
00984             input_error (_("Bad P parameter"));
00985             goto exit_on_error;
00986         }
00987     }
00988     else if (!strcmp (buffer, LABEL_TAXICAB))
00989         input->norm = ERROR_NORM_TAXICAB;
00990     else
00991     {
00992         input_error (_("Unknown error norm"));
00993         goto exit_on_error;
00994     }
00995 }
00996 else
00997     input->norm = ERROR_NORM_EUCLIDIAN;
00998
00999 // Closing the JSON document
01000 g_object_unref (parser);
01001
01002 #if DEBUG_INPUT
01003 fprintf (stderr, "input_open_json: end\n");
01004 #endif
01005 return 1;
01006
01007 exit_on_error:
01008 g_object_unref (parser);
01009 #if DEBUG_INPUT
01010 fprintf (stderr, "input_open_json: end\n");
01011 #endif
01012 return 0;
01013 }
01014
01020 int
01021 input_open (char *filename)
01022 {
01023     xmlDoc *doc;
01024     JsonParser *parser;
01025
01026 #if DEBUG_INPUT
01027 fprintf (stderr, "input_open: start\n");
01028 #endif
01029
01030 // Resetting input data
01031 input_new ();
01032
01033 // Opening input file
01034 #if DEBUG_INPUT
01035 fprintf (stderr, "input_open: opening the input file %s\n", filename);
01036 fprintf (stderr, "input_open: trying XML format\n");
01037 #endif
01038 doc = xmlParseFile (filename);
01039 if (!doc)
01040 {
01041 #if DEBUG_INPUT
01042     fprintf (stderr, "input_open: trying JSON format\n");
01043 #endif
01044     parser = json_parser_new ();
01045     if (!json_parser_load_from_file (parser, filename, NULL))
01046     {
01047         input_error (_("Unable to parse the input file"));
01048         goto exit_on_error;
01049     }
01050     if (!input_open_json (parser))
01051         goto exit_on_error;
01052 }
01053 else if (!input_open_xml (doc))

```

```

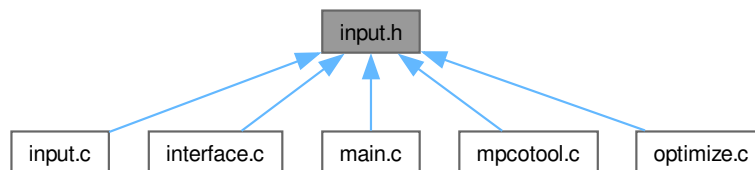
01054     goto exit_on_error;
01055
01056     // Getting the working directory
01057     input->directory = g_path_get_dirname (filename);
01058     input->name = g_path_get_basename (filename);
01059
01060 #if DEBUG_INPUT
01061     fprintf (stderr, "input_open: end\n");
01062 #endif
01063     return 1;
01064
01065 exit_on_error:
01066     jb_show_error (error_message);
01067     g_free (error_message);
01068     input_free ();
01069 #if DEBUG_INPUT
01070     fprintf (stderr, "input_open: end\n");
01071 #endif
01072     return 0;
01073 }

```

## 4.9 input.h File Reference

Header file to define the input functions.

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



### Data Structures

- struct [Input](#)  
*Struct to define the optimization input file.*

### Enumerations

- enum [ClimbingMethod](#) { [CLIMBING\\_METHOD\\_COORDINATES](#) = 0 , [CLIMBING\\_METHOD\\_RANDOM](#) = 1 }  
*Enum to define the methods to estimate the hill climbing.*
- enum [ErrorNorm](#) { [ERROR\\_NORM\\_EUCLIDIAN](#) = 0 , [ERROR\\_NORM\\_MAXIMUM](#) = 1 , [ERROR\\_NORM\\_P](#) = 2 , [ERROR\\_NORM\\_TAXICAB](#) = 3 }  
*Enum to define the error norm.*

### Functions

- void [input\\_new](#) ()
- void [input\\_free](#) ()
- int [input\\_open](#) (char \*filename)

## Variables

- `Input input [1]`  
*Global `Input` struct to set the input data.*
- `const char * result_name`  
*Name of the result file.*
- `const char * variables_name`  
*Name of the variables file.*

## 4.9.1 Detailed Description

Header file to define the input functions.

### Authors

Javier Burguete.

### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file `input.h`.

## 4.9.2 Enumeration Type Documentation

### 4.9.2.1 ClimbingMethod

enum `ClimbingMethod`

Enum to define the methods to estimate the hill climbing.

#### Enumerator

CLIMBING_METHOD_COORDINATES	Coordinates hill climbing method.
CLIMBING_METHOD_RANDOM	Random hill climbing method.

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

```
00043 {  
00044     CLIMBING_METHOD_COORDINATES = 0,  
00045     CLIMBING_METHOD_RANDOM = 1,  
00046 };
```

### 4.9.2.2 ErrorNorm

enum `ErrorNorm`

Enum to define the error norm.

## Enumerator

ERROR_NORM_EUCLIDIAN	Euclidian norm: $\sqrt{\sum_i (w_i x_i)^2}$ .
ERROR_NORM_MAXIMUM	Maximum norm: $\max_i  w_i x_i $ .
ERROR_NORM_P	P-norm $\sqrt[p]{\sum_i  w_i x_i ^p}$ .
ERROR_NORM_TAXICAB	Taxicab norm $\sum_i  w_i x_i $ .

Definition at line 49 of file [input.h](#).

```
00050 {
00051     ERROR_NORM_EUCLIDIAN = 0,
00052     ERROR_NORM_MAXIMUM = 1,
00053     ERROR_NORM_P = 2,
00054     ERROR_NORM_TAXICAB = 3
00055 };
```

## 4.9.3 Function Documentation

### 4.9.3.1 input\_free()

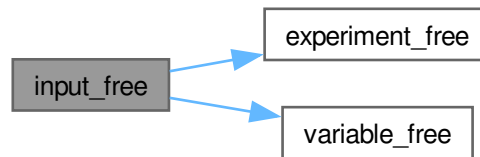
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 86 of file [input.c](#).

```
00087 {
00088     unsigned int i;
00089     #if DEBUG_INPUT
00090     fprintf (stderr, "input_free: start\n");
00091     #endif
00092     g_free (input->name);
00093     g_free (input->directory);
00094     for (i = 0; i < input->nexperiments; ++i)
00095         experiment_free (input->experiment + i, input->type);
00096     for (i = 0; i < input->nvariables; ++i)
00097         variable_free (input->variable + i, input->type);
00098     g_free (input->experiment);
00099     g_free (input->variable);
00100     if (input->type == INPUT_TYPE_XML)
00101     {
00102         xmlFree (input->cleaner);
00103         xmlFree (input->evaluator);
00104         xmlFree (input->simulator);
00105         xmlFree (input->result);
00106         xmlFree (input->variables);
00107     }
00108     else
00109     {
00110         g_free (input->cleaner);
00111         g_free (input->evaluator);
00112         g_free (input->simulator);
00113         g_free (input->result);
00114         g_free (input->variables);
00115     }
00116     input->nexperiments = input->nvariables = input->nsteps
00117         = input->nfinal_steps = 0;
00118     #if DEBUG_INPUT
00119     fprintf (stderr, "input_free: end\n");
00120     #endif
00121 }
```

Here is the call graph for this function:



#### 4.9.3.2 input\_new()

```
void input_new ( )
```

Function to create a new [Input](#) struct.

Definition at line 66 of file [input.c](#).

```

00067 {
00068     #if DEBUG_INPUT
00069         fprintf (stderr, "input_new: start\n");
00070     #endif
00071     input->nvariables = input->nexperiments = input->nsteps = input->nfinal_steps
00072         = 0;
00073     input->simulator = input->evaluator = input->cleaner = input->directory
00074         = input->name = NULL;
00075     input->experiment = NULL;
00076     input->variable = NULL;
00077     #if DEBUG_INPUT
00078         fprintf (stderr, "input_new: end\n");
00079     #endif
00080 }
```

#### 4.9.3.3 input\_open()

```
int input_open (
    char * filename )
```

Function to open the input file.

##### Returns

1\_on\_success, 0\_on\_error.

##### Parameters

<i>filename</i>	<a href="#">Input</a> data file name.
-----------------	---------------------------------------

Definition at line 1021 of file [input.c](#).

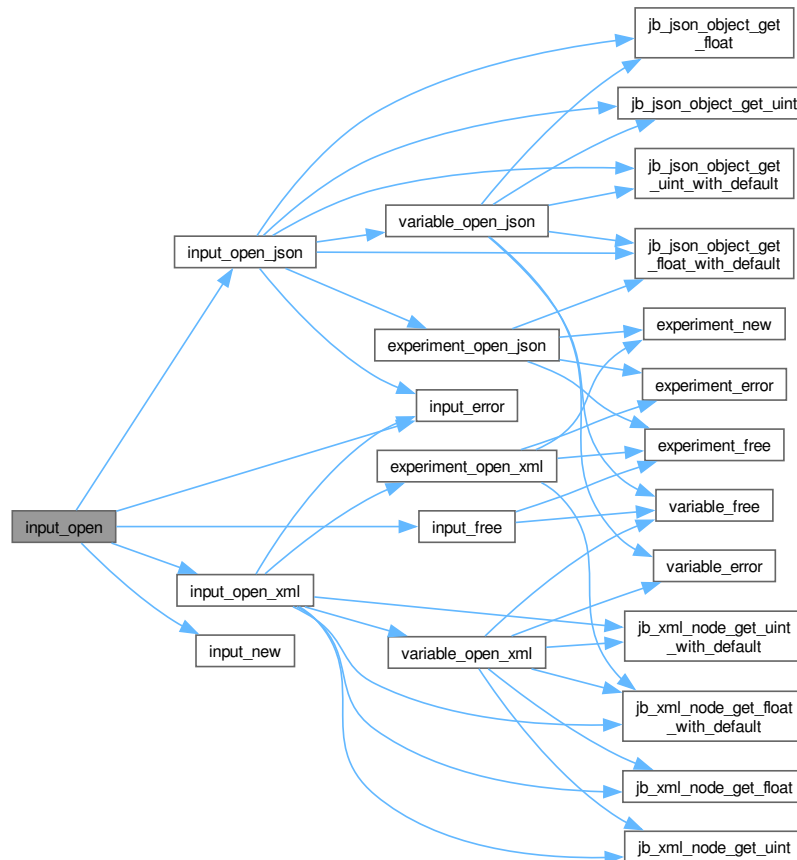
```

01022 {
01023     xmlDoc *doc;
01024     JsonParser *parser;
```



```
01025
01026 #if DEBUG_INPUT
01027     fprintf (stderr, "input_open: start\n");
01028 #endif
01029
01030     // Resetting input data
01031     input_new ();
01032
01033     // Opening input file
01034 #if DEBUG_INPUT
01035     fprintf (stderr, "input_open: opening the input file %s\n", filename);
01036     fprintf (stderr, "input_open: trying XML format\n");
01037 #endif
01038     doc = xmlParseFile (filename);
01039     if (!doc)
01040     {
01041         #if DEBUG_INPUT
01042             fprintf (stderr, "input_open: trying JSON format\n");
01043         #endif
01044         parser = json_parser_new ();
01045         if (!json_parser_load_from_file (parser, filename, NULL))
01046         {
01047             input_error (_("Unable to parse the input file"));
01048             goto exit_on_error;
01049         }
01050         if (!input_open_json (parser))
01051             goto exit_on_error;
01052     }
01053     else if (!input_open_xml (doc))
01054         goto exit_on_error;
01055
01056     // Getting the working directory
01057     input->directory = g_path_get_dirname (filename);
01058     input->name = g_path_get_basename (filename);
01059
01060 #if DEBUG_INPUT
01061     fprintf (stderr, "input_open: end\n");
01062 #endif
01063     return 1;
01064
01065 exit_on_error:
01066     jb_show_error (error_message);
01067     g_free (error_message);
01068     input_free ();
01069 #if DEBUG_INPUT
01070     fprintf (stderr, "input_open: end\n");
01071 #endif
01072     return 0;
01073 }
```

Here is the call graph for this function:



## 4.9.4 Variable Documentation

### 4.9.4.1 input

```
Input input[1] [extern]
```

Global [Input](#) struct to set the input data.

Definition at line 57 of file [input.c](#).

### 4.9.4.2 result\_name

```
const char* result_name [extern]
```

Name of the result file.

Definition at line 59 of file [input.c](#).

#### 4.9.4.3 variables\_name

```
const char* variables_name [extern]
```

Name of the variables file.

Definition at line 60 of file [input.c](#).

## 4.10 input.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INPUT__H
00033 #define INPUT__H 1
00034
00035 enum ClimbingMethod
00036 {
00037     CLIMBING_METHOD_COORDINATES = 0,
00038     CLIMBING_METHOD_RANDOM = 1,
00039 };
00040
00041 enum ErrorNorm
00042 {
00043     ERROR_NORM_EUCLIDIAN = 0,
00044     ERROR_NORM_MAXIMUM = 1,
00045     ERROR_NORM_P = 2,
00046     ERROR_NORM_TAXICAB = 3
00047 };
00048
00049 typedef struct
00050 {
00051     Experiment *experiment;
00052     Variable *variable;
00053     char *result;
00054     char *variables;
00055     char *simulator;
00056     char *evaluator;
00057     char *cleaner;
00058     char *directory;
00059     char *name;
00060     double tolerance;
00061     double mutation_ratio;
00062     double reproduction_ratio;
00063     double adaptation_ratio;
00064     double relaxation;
00065     double p;
00066     double threshold;
```

```

00084 unsigned long int seed;
00086 unsigned int nvariables;
00087 unsigned int nexperiments;
00088 unsigned int nsimulations;
00089 unsigned int algorithm;
00090 unsigned int nsteps;
00092 unsigned int nfinal_steps;
00094 unsigned int climbing;
00095 unsigned int nestimates;
00097 unsigned int niterations;
00098 unsigned int nbest;
00099 unsigned int norm;
00100 unsigned int type;
00101 unsigned int template_flags;
00102 } Input;
00103
00104 extern Input input[1];
00105 extern const char *result_name;
00106 extern const char *variables_name;
00107
00108 // Public functions
00109 void input_new ();
00110 void input_free ();
00111 int input_open (char *filename);
00112
00113 #endif

```

## 4.11 interface.c File Reference

Source file to define the graphical interface functions.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "jb/src/jb_xml.h"
#include "jb/src/jb_json.h"
#include "jb/src/jb_win.h"
#include "genetic/genetic.h"
#include "tools.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"

```

Include dependency graph for interface.c:



## Macros

- #define `DEBUG_INTERFACE` 1  
*Macro to debug interface functions.*
- #define `INPUT_FILE` "test-ga.xml"  
*Macro to define the initial input file.*

## Functions

- static void `input_save_climbing_xml` (xmlNode \*node)
- static void `input_save_climbing_json` (JsonNode \*node)
- static void `input_save_xml` (xmlDoc \*doc)
- static void `input_save_json` (JsonGenerator \*generator)
- static void `input_save` (char \*filename)
- static void `dialog_options_close` (GtkDialog \*dlg, int response\_id)
- static void `options_new` ()
- static void `running_new` ()
- static unsigned int `window_get_algorithm` ()
- static unsigned int `window_get_climbing` ()
- static unsigned int `window_get_norm` ()
- static void `window_save_climbing` ()
- static void `dialog_save_close` (GtkFileChooserDialog \*dlg, int response\_id)
- static void `window_save` ()
- static void `window_run` ()
- static void `window_help` ()
- static void `window_about` ()
- static void `window_update_climbing` ()
- static void `window_update` ()
- static void `window_set_algorithm` ()
- static void `window_set_experiment` ()
- static void `window_remove_experiment` ()
- static void `window_add_experiment` ()
- static void `dialog_name_experiment_close` (GtkFileChooserDialog \*dlg, int response\_id, void \*data)
- static void `window_name_experiment` ()
- static void `window_weight_experiment` ()
- static void `window_inputs_experiment` ()
- static void `window_template_experiment_close` (GtkFileChooserDialog \*dlg, int response\_id, void \*data)
- static void `window_template_experiment` (void \*data)
- static void `window_set_variable` ()
- static void `window_remove_variable` ()
- static void `window_add_variable` ()
- static void `window_label_variable` ()
- static void `window_precision_variable` ()
- static void `window_rangemin_variable` ()
- static void `window_rangemax_variable` ()
- static void `window_rangeminabs_variable` ()
- static void `window_rangemaxabs_variable` ()
- static void `window_step_variable` ()
- static void `window_update_variable` ()
- static int `window_read` (char \*filename)
- static void `dialog_open_close` (GtkFileChooserDialog \*dlg, int response\_id)
- static void `window_open` ()
- static void `dialog_simulator_close` (GtkFileChooserDialog \*dlg, int response\_id)
- static void `dialog_simulator` ()
- static void `dialog_evaluator_close` (GtkFileChooserDialog \*dlg, int response\_id)
- static void `dialog_evaluator` ()
- static void `dialog_cleaner_close` (GtkFileChooserDialog \*dlg, int response\_id)
- static void `dialog_cleaner` ()
- void `window_new` (GtkApplication \*application)

## Variables

- [Window window](#) [1]  
*Window struct to define the main interface window.*
- static const char \* [logo](#) []  
*Logo pixmap.*
- static [Options options](#) [1]  
*Options struct to define the options dialog.*
- static [Running running](#) [1]  
*Running struct to define the running dialog.*

### 4.11.1 Detailed Description

Source file to define the graphical interface functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [interface.c](#).

### 4.11.2 Macro Definition Documentation

#### 4.11.2.1 DEBUG\_INTERFACE

```
#define DEBUG_INTERFACE 1
```

Macro to debug interface functions.

Definition at line 69 of file [interface.c](#).

#### 4.11.2.2 INPUT\_FILE

```
#define INPUT_FILE "test-ga.xml"
```

Macro to define the initial input file.

Definition at line 78 of file [interface.c](#).

### 4.11.3 Function Documentation

#### 4.11.3.1 dialog\_cleaner()

```
static void dialog_cleaner ( ) [static]
```

Function to open a dialog to save the cleaner file.

Definition at line 2340 of file [interface.c](#).

```
02341 {
02342     GtkFileChooserDialog *dlg;
02343     #if DEBUG_INTERFACE
02344         fprintf (stderr, "dialog_cleaner: start\n");
02345     #endif
02346     dlg = (GtkFileChooserDialog *)
02347         gtk_file_chooser_dialog_new (_("Open cleaner file"),
02348                                     window->window,
02349                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02350                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02351                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02352     g_signal_connect (dlg, "response", G_CALLBACK (dialog_cleaner_close), NULL);
02353     gtk_window_present (GTK_WINDOW (dlg));
02354     #if DEBUG_INTERFACE
02355         fprintf (stderr, "dialog_cleaner: end\n");
02356     #endif
02357 }
```

Here is the call graph for this function:



#### 4.11.3.2 dialog\_cleaner\_close()

```
static void dialog_cleaner_close (
    GtkFileChooserDialog * dlg,
    int response_id ) [static]
```

Function to save the close the cleaner file dialog.

##### Parameters

<i>dlg</i>	GtkFileChooserDialog dialog.
<i>response_id</i>	Response identifier.

Definition at line 2309 of file [interface.c](#).

```
02312 {
02313     GFile *file1, *file2;
02314     char *buffer1, *buffer2;
02315     #if DEBUG_INTERFACE
02316         fprintf (stderr, "dialog_cleaner_close: start\n");
02317     #endif
```

```

02318     if (response_id == GTK_RESPONSE_OK)
02319     {
02320         buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02321         file1 = g_file_new_for_path (buffer1);
02322         file2 = g_file_new_for_path (input->directory);
02323         buffer2 = g_file_get_relative_path (file2, file1);
02324         input->cleaner = g_strdup (buffer2);
02325         g_free (buffer2);
02326         g_object_unref (file2);
02327         g_object_unref (file1);
02328         g_free (buffer1);
02329     }
02330     gtk_window_destroy (GTK_WINDOW (dlg));
02331     #if DEBUG_INTERFACE
02332     fprintf (stderr, "dialog_cleaner_close: end\n");
02333     #endif
02334 }

```

#### 4.11.3.3 dialog\_evaluator()

```
static void dialog_evaluator ( ) [static]
```

Function to open a dialog to save the evaluator file.

Definition at line 2286 of file [interface.c](#).

```

02287 {
02288     GtkFileChooserDialog *dlg;
02289     #if DEBUG_INTERFACE
02290     fprintf (stderr, "dialog_evaluator: start\n");
02291     #endif
02292     dlg = (GtkFileChooserDialog *)
02293         gtk_file_chooser_dialog_new (_("Open evaluator file"),
02294                                     window->window,
02295                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02296                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02297                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02298     g_signal_connect (dlg, "response", G_CALLBACK (dialog_evaluator_close), NULL);
02299     gtk_window_present (GTK_WINDOW (dlg));
02300     #if DEBUG_INTERFACE
02301     fprintf (stderr, "dialog_evaluator: end\n");
02302     #endif
02303 }

```

Here is the call graph for this function:



#### 4.11.3.4 dialog\_evaluator\_close()

```
static void dialog_evaluator_close (
    GtkFileChooserDialog * dlg,
    int response_id ) [static]
```

Function to save the close the evaluator file dialog.



## Parameters

<i>dlg</i>	GtkFileChooserDialog dialog.
<i>response</i> ↔ <i>_id</i>	Response identifier.

Definition at line 2255 of file [interface.c](#).

```

02258 {
02259     GFile *file1, *file2;
02260     char *buffer1, *buffer2;
02261     #if DEBUG_INTERFACE
02262     fprintf (stderr, "dialog_evaluator_close: start\n");
02263     #endif
02264     if (response_id == GTK_RESPONSE_OK)
02265     {
02266         buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02267         file1 = g_file_new_for_path (buffer1);
02268         file2 = g_file_new_for_path (input->directory);
02269         buffer2 = g_file_get_relative_path (file2, file1);
02270         input->evaluator = g_strdup (buffer2);
02271         g_free (buffer2);
02272         g_object_unref (file2);
02273         g_object_unref (file1);
02274         g_free (buffer1);
02275     }
02276     gtk_window_destroy (GTK_WINDOW (dlg));
02277     #if DEBUG_INTERFACE
02278     fprintf (stderr, "dialog_evaluator_close: end\n");
02279     #endif
02280 }
```

## 4.11.3.5 dialog\_name\_experiment\_close()

```

static void dialog_name_experiment_close (
    GtkFileChooserDialog * dlg,
    int response_id,
    void * data ) [static]
```

Function to close the experiment name dialog.

## Parameters

<i>dlg</i>	GtkFileChooserDialog struct.
<i>response</i> ↔ <i>_id</i>	Response identifier.
<i>data</i>	Function data.

Definition at line 1487 of file [interface.c](#).

```

01491 {
01492     char *buffer;
01493     unsigned int i;
01494     #if DEBUG_INTERFACE
01495     fprintf (stderr, "window_name_experiment_close: start\n");
01496     #endif
01497     i = (size_t) data;
01498     if (response_id == GTK_RESPONSE_OK)
01499     {
01500         buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01501         g_signal_handler_block (window->combo_experiment, window->id_experiment);
01502         gtk_combo_box_text_remove (window->combo_experiment, i);
01503         gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01504         gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01505         g_signal_handler_unblock (window->combo_experiment,
01506                                 window->id_experiment);
01507         g_free (buffer);
01508     }
01509     #if DEBUG_INTERFACE
```

```

01510     fprintf (stderr, "window_name_experiment_close: end\n");
01511 #endif
01512 }

```

#### 4.11.3.6 dialog\_open\_close()

```

static void dialog_open_close (
    GtkFileChooserDialog * dlg,
    int response_id ) [static]

```

Function to close the input data dialog.

##### Parameters

<i>dlg</i>	GtkFileChooserDialog dialog.
<i>response_id</i>	Response identifier.

Definition at line 2089 of file [interface.c](#).

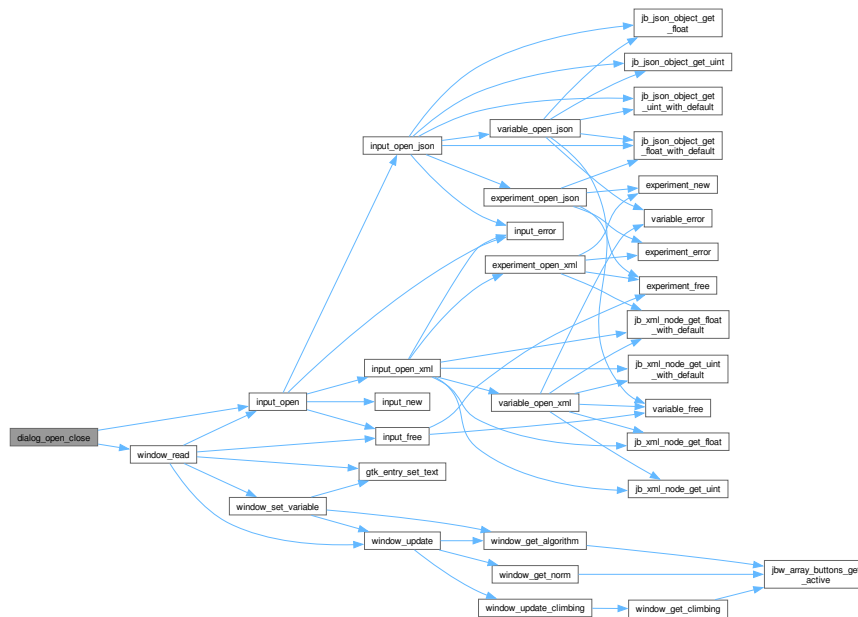
```

02092 {
02093     char *buffer, *directory, *name;
02094     GFile *file;
02095
02096 #if DEBUG_INTERFACE
02097     fprintf (stderr, "dialog_open_close: start\n");
02098 #endif
02099
02100     // Saving a backup of the current input file
02101     directory = g_strdup (input->directory);
02102     name = g_strdup (input->name);
02103
02104     // If OK saving
02105     if (response_id == GTK_RESPONSE_OK)
02106     {
02107
02108         // Trying to open the input file
02109         file = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (dlg));
02110         buffer = g_file_get_path (file);
02111 #if DEBUG_INTERFACE
02112         fprintf (stderr, "dialog_open_close: file name=%s\n", buffer);
02113 #endif
02114         if (!window_read (buffer))
02115         {
02116 #if DEBUG_INTERFACE
02117             fprintf (stderr, "dialog_open_close: error reading input file\n");
02118 #endif
02119             g_free (buffer);
02120
02121             // Reading backup file on error
02122             buffer = g_build_filename (directory, name, NULL);
02123             input->result = input->variables = NULL;
02124             if (!input_open (buffer))
02125             {
02126
02127                 // Closing on backup file reading error
02128 #if DEBUG_INTERFACE
02129                 fprintf (stderr,
02130                     "dialog_open_close: error reading backup file\n");
02131 #endif
02132             }
02133             g_free (buffer);
02134             g_object_unref (file);
02135         }
02136     }
02137
02138     // Freeing and closing
02139     g_free (name);
02140     g_free (directory);
02141     gtk_window_destroy (GTK_WINDOW (dlg));
02142
02143 #if DEBUG_INTERFACE
02144     fprintf (stderr, "dialog_open_close: end\n");
02145 #endif
02146 }

```

```
02147 }
```

Here is the call graph for this function:



#### 4.11.3.7 dialog\_options\_close()

```
static void dialog_options_close (
    GtkWidget * dlg,
    int response_id ) [static]
```

Function to close the options dialog.

##### Parameters

<i>dlg</i>	GtkWidget options dialog.
<i>response_id</i>	Response identifier.

Definition at line 656 of file [interface.c](#).

```

00658 {
00659     #if DEBUG_INTERFACE
00660         fprintf (stderr, "dialog_options_close: start\n");
00661     #endif
00662     if (response_id == GTK_RESPONSE_OK)
00663     {
00664         input->seed
00665             = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00666         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00667         nthreads_climbing
00668             = gtk_spin_button_get_value_as_int (options->spin_climbing);
00669     }
00670     gtk_window_destroy (GTK_WINDOW (dlg));
00671     #if DEBUG_INTERFACE
00672         fprintf (stderr, "dialog_options_close: end\n");
00673     #endif
00674 }
```

#### 4.11.3.8 dialog\_save\_close()

```
static void dialog_save_close (
    GtkFileChooserDialog * dlg,
    int response_id ) [static]
```

Function to close the save dialog.

##### Parameters

<i>dlg</i>	GtkFileChooserDialog dialog.
<i>response_id</i>	Response identifier.

Definition at line 872 of file [interface.c](#).

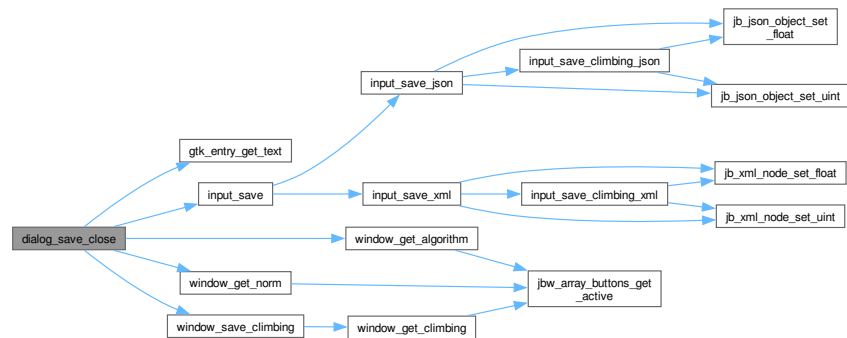
```
00875 {
00876     GtkFileFilter *filter1;
00877     char *buffer;
00878     #if DEBUG_INTERFACE
00879     fprintf (stderr, "dialog_save_close: start\n");
00880     #endif
00881     // If OK response then saving
00882     if (response_id == GTK_RESPONSE_OK)
00883     {
00884         // Setting input file type
00885         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00886         buffer = (char *) gtk_file_filter_get_name (filter1);
00887         if (!strcmp (buffer, "XML"))
00888             input->type = INPUT_TYPE_XML;
00889         else
00890             input->type = INPUT_TYPE_JSON;
00891
00892         // Adding properties to the root XML node
00893         input->simulator
00894         = g_strdup (gtk_button_get_label (window->button_simulator));
00895         if (gtk_check_button_get_active (window->check_evaluator))
00896             input->evaluator
00897             = g_strdup (gtk_button_get_label (window->button_evaluator));
00898         else
00899             input->evaluator = NULL;
00900         if (gtk_check_button_get_active (window->check_cleaner))
00901             input->cleaner
00902             = g_strdup (gtk_button_get_label (window->button_cleaner));
00903         else
00904             input->cleaner = NULL;
00905         if (input->type == INPUT_TYPE_XML)
00906         {
00907             input->result
00908             = (char *) xmlStrdup ((const xmlChar *)
00909                                   gtk_entry_get_text (window->entry_result));
00910             input->variables
00911             = (char *) xmlStrdup ((const xmlChar *)
00912                                   gtk_entry_get_text (window->entry_variables));
00913         }
00914         else
00915         {
00916             input->result = g_strdup (gtk_entry_get_text (window->entry_result));
00917             input->variables =
00918             g_strdup (gtk_entry_get_text (window->entry_variables));
00919         }
00920
00921         // Setting the algorithm
00922         switch (window_get_algorithm ())
00923         {
00924             case ALGORITHM_MONTE_CARLO:
00925                 input->algorithm = ALGORITHM_MONTE_CARLO;
00926                 input->nsimulations
00927                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00928                 input->niterations
00929                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00930                 input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00931                 input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00932                 window_save_climbing ();
00933                 break;
00934             case ALGORITHM_SWEEP:
00935                 input->algorithm = ALGORITHM_SWEEP;
00936                 input->niterations
```

```

00937         = gtk_spin_button_get_value_as_int (window->spin_iterations);
00938     input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00939     input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00940     window_save_climbing ();
00941     break;
00942 case ALGORITHM_ORTHOGONAL:
00943     input->algorithm = ALGORITHM_ORTHOGONAL;
00944     input->niterations
00945         = gtk_spin_button_get_value_as_int (window->spin_iterations);
00946     input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00947     input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00948     window_save_climbing ();
00949     break;
00950 default:
00951     input->algorithm = ALGORITHM_GENETIC;
00952     input->nsimulations
00953         = gtk_spin_button_get_value_as_int (window->spin_population);
00954     input->niterations
00955         = gtk_spin_button_get_value_as_int (window->spin_generations);
00956     input->mutation_ratio
00957         = gtk_spin_button_get_value (window->spin_mutation);
00958     input->reproduction_ratio
00959         = gtk_spin_button_get_value (window->spin_reproduction);
00960     input->adaptation_ratio
00961         = gtk_spin_button_get_value (window->spin_adaptation);
00962 }
00963 input->norm = window_get_norm ();
00964 input->p = gtk_spin_button_get_value (window->spin_p);
00965 input->threshold = gtk_spin_button_get_value (window->spin_threshold);
00966
00967 // Saving the XML file
00968 buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
00969 input_save (buffer);
00970
00971 // Closing and freeing memory
00972 g_free (buffer);
00973 }
00974 gtk_window_destroy (GTK_WINDOW (dlg));
00975 #if DEBUG_INTERFACE
00976 fprintf (stderr, "dialog_save_close: end\n");
00977 #endif
00978 }

```

Here is the call graph for this function:



#### 4.11.3.9 dialog\_simulator()

```
static void dialog_simulator ( ) [static]
```

Function to open a dialog to save the simulator file.

Definition at line 2232 of file [interface.c](#).

```

02233 {
02234     GtkFileChooserDialog *dlg;
02235     #if DEBUG_INTERFACE

```

```

02236     fprintf (stderr, "dialog_simulator: start\n");
02237 #endif
02238     dlg = (GtkFileChooserDialog *)
02239         gtk_file_chooser_dialog_new (_("Open simulator file"),
02240                                     window->window,
02241                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02242                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02243                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02244     g_signal_connect (dlg, "response", G_CALLBACK (dialog_simulator_close), NULL);
02245     gtk_window_present (GTK_WINDOW (dlg));
02246 #if DEBUG_INTERFACE
02247     fprintf (stderr, "dialog_simulator: end\n");
02248 #endif
02249 }

```

Here is the call graph for this function:



#### 4.11.3.10 dialog\_simulator\_close()

```

static void dialog_simulator_close (
    GtkFileChooserDialog * dlg,
    int response_id ) [static]

```

Function to save the close the simulator file dialog.

##### Parameters

<i>dlg</i>	GtkFileChooserDialog dialog.
<i>response_id</i>	Response identifier.

Definition at line 2201 of file [interface.c](#).

```

02204 {
02205     GFile *file1, *file2;
02206     char *buffer1, *buffer2;
02207 #if DEBUG_INTERFACE
02208     fprintf (stderr, "dialog_simulator_close: start\n");
02209 #endif
02210     if (response_id == GTK_RESPONSE_OK)
02211     {
02212         buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02213         file1 = g_file_new_for_path (buffer1);
02214         file2 = g_file_new_for_path (input->directory);
02215         buffer2 = g_file_get_relative_path (file2, file1);
02216         input->simulator = g_strdup (buffer2);
02217         g_free (buffer2);
02218         g_object_unref (file2);
02219         g_object_unref (file1);
02220         g_free (buffer1);
02221     }
02222     gtk_window_destroy (GTK_WINDOW (dlg));
02223 #if DEBUG_INTERFACE
02224     fprintf (stderr, "dialog_simulator_close: end\n");
02225 #endif
02226 }

```

## 4.11.3.11 input\_save()

```
static void input_save (
    char * filename ) [inline], [static]
```

Function to save the input file.

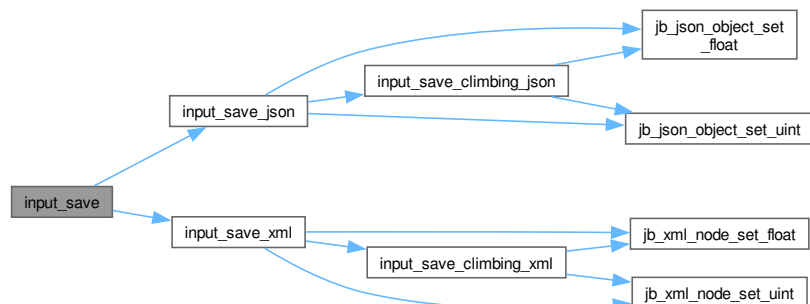
## Parameters

<i>filename</i>	Input file name.
-----------------	------------------

Definition at line 608 of file [interface.c](#).

```
00609 {
00610     xmlDoc *doc;
00611     JsonGenerator *generator;
00612
00613     #if DEBUG_INTERFACE
00614         fprintf (stderr, "input_save: start\n");
00615     #endif
00616
00617     // Getting the input file directory
00618     input->name = g_path_get_basename (filename);
00619     input->directory = g_path_get_dirname (filename);
00620
00621     if (input->type == INPUT_TYPE_XML)
00622     {
00623         // Opening the input file
00624         doc = xmlNewDoc ((const xmlChar *) "1.0");
00625         input_save_xml (doc);
00626
00627         // Saving the XML file
00628         xmlSaveFormatFile (filename, doc, 1);
00629
00630         // Freeing memory
00631         xmlFreeDoc (doc);
00632     }
00633     else
00634     {
00635         // Opening the input file
00636         generator = json_generator_new ();
00637         json_generator_set_pretty (generator, TRUE);
00638         input_save_json (generator);
00639
00640         // Saving the JSON file
00641         json_generator_to_file (generator, filename, NULL);
00642
00643         // Freeing memory
00644         g_object_unref (generator);
00645     }
00646
00647     #if DEBUG_INTERFACE
00648         fprintf (stderr, "input_save: end\n");
00649     #endif
00650 }
```

Here is the call graph for this function:



#### 4.11.3.12 input\_save\_climbing\_json()

```
static void input_save_climbing_json (
    JsonNode * node ) [static]
```

Function to save the hill climbing method data in a JSON node.

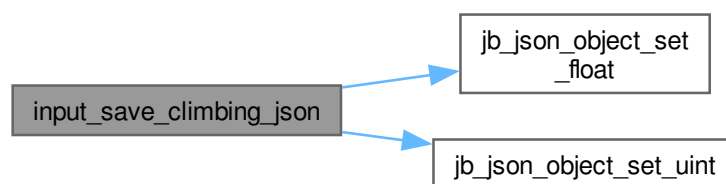
##### Parameters

<i>node</i>	JSON node.
-------------	------------

Definition at line 206 of file [interface.c](#).

```
00207 {
00208     JsonObject *object;
00209     #if DEBUG_INTERFACE
00210     fprintf (stderr, "input_save_climbing_json: start\n");
00211     #endif
00212     object = json_node_get_object (node);
00213     if (input->nsteps)
00214     {
00215         jb_json_object_set_uint (object, LABEL_NSTEPS, input->nsteps);
00216         if (input->relaxation != DEFAULT_RELAXATION)
00217             jb_json_object_set_float (object, LABEL_RELAXATION, input->relaxation);
00218         switch (input->climbing)
00219         {
00220             case CLIMBING_METHOD_COORDINATES:
00221                 json_object_set_string_member (object, LABEL_CLIMBING,
00222                                                 LABEL_COORDINATES);
00223                 break;
00224             default:
00225                 json_object_set_string_member (object, LABEL_CLIMBING, LABEL_RANDOM);
00226                 jb_json_object_set_uint (object, LABEL_NESTIMATES, input->nestimates);
00227         }
00228     }
00229     #if DEBUG_INTERFACE
00230     fprintf (stderr, "input_save_climbing_json: end\n");
00231     #endif
00232 }
```

Here is the call graph for this function:



#### 4.11.3.13 input\_save\_climbing\_xml()

```
static void input_save_climbing_xml (
    xmlNode * node ) [static]
```

Function to save the hill climbing method data in a XML node.



## Parameters

<i>node</i>	XML node.
-------------	-----------

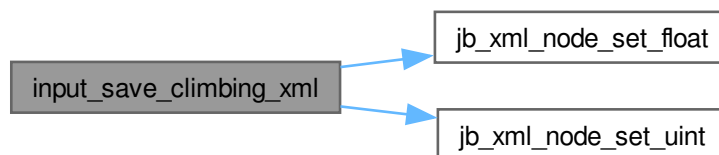
Definition at line 172 of file [interface.c](#).

```

00173 {
00174     #if DEBUG_INTERFACE
00175     fprintf (stderr, "input_save_climbing_xml: start\n");
00176     #endif
00177     if (input->nsteps)
00178     {
00179         jb_xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00180                             input->nsteps);
00181         if (input->relaxation != DEFAULT_RELAXATION)
00182             jb_xml_node_set_float (node, (const xmlChar *) LABEL_RELAXATION,
00183                                   input->relaxation);
00184         switch (input->climbing)
00185         {
00186             case CLIMBING_METHOD_COORDINATES:
00187                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00188                             (const xmlChar *) LABEL_COORDINATES);
00189                 break;
00190             default:
00191                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00192                             (const xmlChar *) LABEL_RANDOM);
00193                 jb_xml_node_set_uint (node, (const xmlChar *) LABEL_NESTIMATES,
00194                                     input->nestimates);
00195         }
00196     }
00197     #if DEBUG_INTERFACE
00198     fprintf (stderr, "input_save_climbing_xml: end\n");
00199     #endif
00200 }

```

Here is the call graph for this function:



#### 4.11.3.14 input\_save\_json()

```

static void input_save_json (
    JsonGenerator * generator ) [inline], [static]

```

Function to save the input file in JSON format.

## Parameters

<i>generator</i>	JsonGenerator struct.
------------------	-----------------------

Definition at line 427 of file [interface.c](#).

```

00428 {

```

```

00429 unsigned int i, j;
00430 char *buffer;
00431 JsonNode *node, *child;
00432 JsonObject *object;
00433 JsonArray *array;
00434 GFile *file, *file2;
00435
00436 #if DEBUG_INTERFACE
00437 fprintf (stderr, "input_save_json: start\n");
00438 #endif
00439
00440 // Setting root JSON node
00441 object = json_object_new ();
00442 node = json_node_new (JSON_NODE_OBJECT);
00443 json_node_set_object (node, object);
00444 json_generator_set_root (generator, node);
00445
00446 // Adding properties to the root JSON node
00447 if (strcmp (input->result, result_name))
00448     json_object_set_string_member (object, LABEL_RESULT_FILE, input->result);
00449 if (strcmp (input->variables, variables_name))
00450     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00451                                   input->variables);
00452 file = g_file_new_for_path (input->directory);
00453 file2 = g_file_new_for_path (input->simulator);
00454 buffer = g_file_get_relative_path (file, file2);
00455 g_object_unref (file2);
00456 json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00457 g_free (buffer);
00458 if (input->evaluator)
00459 {
00460     file2 = g_file_new_for_path (input->evaluator);
00461     buffer = g_file_get_relative_path (file, file2);
00462     g_object_unref (file2);
00463     if (strlen (buffer))
00464         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00465     g_free (buffer);
00466 }
00467 if (input->cleaner)
00468 {
00469     file2 = g_file_new_for_path (input->cleaner);
00470     buffer = g_file_get_relative_path (file, file2);
00471     g_object_unref (file2);
00472     if (strlen (buffer))
00473         json_object_set_string_member (object, LABEL_CLEANER, buffer);
00474     g_free (buffer);
00475 }
00476 if (input->seed != DEFAULT_RANDOM_SEED)
00477     jb_json_object_set_uint (object, LABEL_SEED, input->seed);
00478
00479 // Setting the algorithm
00480 buffer = (char *) g_slice_alloc (64);
00481 switch (input->algorithm)
00482 {
00483     case ALGORITHM_MONTE_CARLO:
00484         json_object_set_string_member (object, LABEL_ALGORITHM,
00485                                       LABEL_MONTE_CARLO);
00486         snprintf (buffer, 64, "%u", input->nsimulations);
00487         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00488         snprintf (buffer, 64, "%u", input->niterations);
00489         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00490         snprintf (buffer, 64, "%.3lg", input->tolerance);
00491         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00492         snprintf (buffer, 64, "%u", input->nbest);
00493         json_object_set_string_member (object, LABEL_NBEST, buffer);
00494         input_save_climbing_json (node);
00495         break;
00496     case ALGORITHM_SWEEP:
00497         json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_SWEEP);
00498         snprintf (buffer, 64, "%u", input->niterations);
00499         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00500         snprintf (buffer, 64, "%.3lg", input->tolerance);
00501         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00502         snprintf (buffer, 64, "%u", input->nbest);
00503         json_object_set_string_member (object, LABEL_NBEST, buffer);
00504         input_save_climbing_json (node);
00505         break;
00506     case ALGORITHM_ORTHOGONAL:
00507         json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_ORTHOGONAL);
00508         snprintf (buffer, 64, "%u", input->niterations);
00509         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00510         snprintf (buffer, 64, "%.3lg", input->tolerance);
00511         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00512         snprintf (buffer, 64, "%u", input->nbest);
00513         json_object_set_string_member (object, LABEL_NBEST, buffer);
00514         input_save_climbing_json (node);
00515         break;

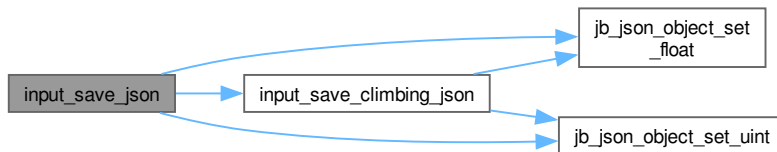
```

```

00516     default:
00517         json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_GENETIC);
00518         snprintf (buffer, 64, "%u", input->nsimulations);
00519         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00520         snprintf (buffer, 64, "%u", input->niterations);
00521         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00522         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00523         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00524         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00525         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00526         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00527         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00528         break;
00529     }
00530     g_slice_free1 (64, buffer);
00531     if (input->threshold != 0.)
00532         jb_json_object_set_float (object, LABEL_THRESHOLD, input->threshold);
00533
00534     // Setting the experimental data
00535     array = json_array_new ();
00536     for (i = 0; i < input->nexperiments; ++i)
00537     {
00538         child = json_node_new (JSON_NODE_OBJECT);
00539         object = json_node_get_object (child);
00540         json_object_set_string_member (object, LABEL_NAME,
00541                                     input->experiment[i].name);
00542         if (input->experiment[i].weight != 1.)
00543             jb_json_object_set_float (object, LABEL_WEIGHT,
00544                                     input->experiment[i].weight);
00545         for (j = 0; j < input->experiment->ninputs; ++j)
00546             json_object_set_string_member (object, stencil[j],
00547                                     input->experiment[i].stencil[j]);
00548         json_array_add_element (array, child);
00549     }
00550     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00551
00552     // Setting the variables data
00553     array = json_array_new ();
00554     for (i = 0; i < input->nvariables; ++i)
00555     {
00556         child = json_node_new (JSON_NODE_OBJECT);
00557         object = json_node_get_object (child);
00558         json_object_set_string_member (object, LABEL_NAME,
00559                                     input->variable[i].name);
00560         jb_json_object_set_float (object, LABEL_MINIMUM,
00561                                 input->variable[i].rangemin);
00562         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00563             jb_json_object_set_float (object, LABEL_ABSOLUTE_MINIMUM,
00564                                     input->variable[i].rangeminabs);
00565         jb_json_object_set_float (object, LABEL_MAXIMUM,
00566                                 input->variable[i].rangemax);
00567         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00568             jb_json_object_set_float (object, LABEL_ABSOLUTE_MAXIMUM,
00569                                     input->variable[i].rangemaxabs);
00570         if (input->variable[i].precision != DEFAULT_PRECISION)
00571             jb_json_object_set_uint (object, LABEL_PRECISION,
00572                                    input->variable[i].precision);
00573         if (input->algorithm == ALGORITHM_SWEEP
00574             || input->algorithm == ALGORITHM_ORTHOGONAL)
00575             jb_json_object_set_uint (object, LABEL_NSWEEPS,
00576                                    input->variable[i].nsweeps);
00577         else if (input->algorithm == ALGORITHM_GENETIC)
00578             jb_json_object_set_uint (object, LABEL_NBITS, input->variable[i].nbits);
00579         if (input->nsteps)
00580             jb_json_object_set_float (object, LABEL_STEP, input->variable[i].step);
00581         json_array_add_element (array, child);
00582     }
00583     json_object_set_array_member (object, LABEL_VARIABLES, array);
00584
00585     // Saving the error norm
00586     switch (input->norm)
00587     {
00588     case ERROR_NORM_MAXIMUM:
00589         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00590         break;
00591     case ERROR_NORM_P:
00592         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00593         jb_json_object_set_float (object, LABEL_P, input->p);
00594         break;
00595     case ERROR_NORM_TAXICAB:
00596         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00597     }
00598
00599     #if DEBUG_INTERFACE
00600     fprintf (stderr, "input_save_json: end\n");
00601     #endif
00602 }

```

Here is the call graph for this function:



#### 4.11.3.15 input\_save\_xml()

```
static void input_save_xml (
    xmlDoc * doc ) [inline], [static]
```

Function to save the input file in XML format.

##### Parameters

<i>doc</i>	xmlDoc struct.
------------	----------------

Definition at line 238 of file [interface.c](#).

```

00239 {
00240     unsigned int i, j;
00241     char *buffer;
00242     xmlNode *node, *child;
00243     GFile *file, *file2;
00244
00245     #if DEBUG_INTERFACE
00246         fprintf (stderr, "input_save_xml: start\n");
00247     #endif
00248
00249     // Setting root XML node
00250     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00251     xmlDocSetRootElement (doc, node);
00252
00253     // Adding properties to the root XML node
00254     if (xmlStrcmp
00255         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00256         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00257                     (xmlChar *) input->result);
00258     if (xmlStrcmp
00259         ((const xmlChar *) input->variables, (const xmlChar *) variables_name))
00260         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00261                     (xmlChar *) input->variables);
00262     file = g_file_new_for_path (input->directory);
00263     file2 = g_file_new_for_path (input->simulator);
00264     buffer = g_file_get_relative_path (file, file2);
00265     g_object_unref (file2);
00266     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00267     g_free (buffer);
00268     if (input->evaluator)
00269     {
00270         file2 = g_file_new_for_path (input->evaluator);
00271         buffer = g_file_get_relative_path (file, file2);
00272         g_object_unref (file2);
00273         if (xmlStrlen ((xmlChar *) buffer))
00274             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00275                         (xmlChar *) buffer);
00276         g_free (buffer);
00277     }
00278     if (input->cleaner)
00279     {
00280         file2 = g_file_new_for_path (input->cleaner);
00281         buffer = g_file_get_relative_path (file, file2);

```

```

00282     g_object_unref (file2);
00283     if (xmlStrlen ((xmlChar *) buffer))
00284         xmlSetProp (node, (const xmlChar *) LABEL_CLEANER, (xmlChar *) buffer);
00285     g_free (buffer);
00286 }
00287 if (input->seed != DEFAULT_RANDOM_SEED)
00288     jb_xml_node_set_uint (node, (const xmlChar *) LABEL_SEED, input->seed);
00289
00290 // Setting the algorithm
00291 buffer = (char *) g_slice_alloc (64);
00292 switch (input->algorithm)
00293 {
00294     case ALGORITHM_MONTE_CARLO:
00295         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00296                     (const xmlChar *) LABEL_MONTE_CARLO);
00297         snprintf (buffer, 64, "%u", input->nsimulations);
00298         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00299                     (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%u", input->niterations);
00301         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00302                     (xmlChar *) buffer);
00303         snprintf (buffer, 64, "%.3lg", input->tolerance);
00304         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00305         snprintf (buffer, 64, "%u", input->nbest);
00306         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00307         input_save_climbing_xml (node);
00308         break;
00309     case ALGORITHM_SWEEP:
00310         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00311                     (const xmlChar *) LABEL_SWEEP);
00312         snprintf (buffer, 64, "%u", input->niterations);
00313         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00314                     (xmlChar *) buffer);
00315         snprintf (buffer, 64, "%.3lg", input->tolerance);
00316         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00317         snprintf (buffer, 64, "%u", input->nbest);
00318         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00319         input_save_climbing_xml (node);
00320         break;
00321     case ALGORITHM_ORTHOGONAL:
00322         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00323                     (const xmlChar *) LABEL_ORTHOGONAL);
00324         snprintf (buffer, 64, "%u", input->niterations);
00325         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00326                     (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%.3lg", input->tolerance);
00328         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00329         snprintf (buffer, 64, "%u", input->nbest);
00330         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00331         input_save_climbing_xml (node);
00332         break;
00333     default:
00334         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00335                     (const xmlChar *) LABEL_GENETIC);
00336         snprintf (buffer, 64, "%u", input->nsimulations);
00337         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00338                     (xmlChar *) buffer);
00339         snprintf (buffer, 64, "%u", input->niterations);
00340         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00341                     (xmlChar *) buffer);
00342         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00343         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00344         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00345         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00346                     (xmlChar *) buffer);
00347         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00348         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00349         break;
00350 }
00351 g_slice_free1 (64, buffer);
00352 if (input->threshold != 0.)
00353     jb_xml_node_set_float (node, (const xmlChar *) LABEL_THRESHOLD,
00354                            input->threshold);
00355
00356 // Setting the experimental data
00357 for (i = 0; i < input->nexperiments; ++i)
00358 {
00359     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00360     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361                 (xmlChar *) input->experiment[i].name);
00362     if (input->experiment[i].weight != 1.)
00363         jb_xml_node_set_float (child, (const xmlChar *) LABEL_WEIGHT,
00364                                input->experiment[i].weight);
00365     for (j = 0; j < input->experiment->ninputs; ++j)
00366         xmlSetProp (child, (const xmlChar *) stencil[j],
00367                     (xmlChar *) input->experiment[i].stencil[j]);
00368 }

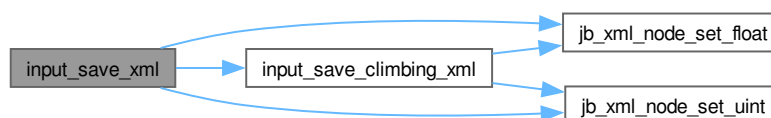
```

```

00369
00370 // Setting the variables data
00371 for (i = 0; i < input->nvariables; ++i)
00372 {
00373     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00374     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00375                 (xmlChar *) input->variable[i].name);
00376     jb_xml_node_set_float (child, (const xmlChar *) LABEL_MINIMUM,
00377                           input->variable[i].rangemin);
00378     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00379         jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM,
00380                               input->variable[i].rangeminabs);
00381     jb_xml_node_set_float (child, (const xmlChar *) LABEL_MAXIMUM,
00382                           input->variable[i].rangemax);
00383     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00384         jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM,
00385                               input->variable[i].rangemaxabs);
00386     if (input->variable[i].precision != DEFAULT_PRECISION)
00387         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_PRECISION,
00388                              input->variable[i].precision);
00389     if (input->algorithm == ALGORITHM_SWEEP
00390         || input->algorithm == ALGORITHM_ORTHOGONAL)
00391         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NSWEEPS,
00392                              input->variable[i].nsweeps);
00393     else if (input->algorithm == ALGORITHM_GENETIC)
00394         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00395                              input->variable[i].nbits);
00396     if (input->nsteps)
00397         jb_xml_node_set_float (child, (const xmlChar *) LABEL_STEP,
00398                               input->variable[i].step);
00399 }
00400
00401 // Saving the error norm
00402 switch (input->norm)
00403 {
00404     case ERROR_NORM_MAXIMUM:
00405         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00406                     (const xmlChar *) LABEL_MAXIMUM);
00407         break;
00408     case ERROR_NORM_P:
00409         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00410                     (const xmlChar *) LABEL_P);
00411         jb_xml_node_set_float (node, (const xmlChar *) LABEL_P, input->p);
00412         break;
00413     case ERROR_NORM_TAXICAB:
00414         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00415                     (const xmlChar *) LABEL_TAXICAB);
00416 }
00417
00418 #if DEBUG_INTERFACE
00419 fprintf (stderr, "input_save: end\n");
00420 #endif
00421 }

```

Here is the call graph for this function:



#### 4.11.3.16 options\_new()

```
static void options_new ( ) [static]
```

Function to open the options dialog.

Definition at line 680 of file [interface.c](#).

```

00681 {
00682 #if DEBUG_INTERFACE
00683     fprintf (stderr, "options_new: start\n");
00684 #endif
00685     options->label_seed = (GtkLabel *)
00686         gtk_label_new (_("Pseudo-random numbers generator seed"));
00687     options->spin_seed = (GtkSpinButton *)
00688         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00689     gtk_widget_set_tooltip_text
00690         (GTK_WIDGET (options->spin_seed),
00691          _("Seed to init the pseudo-random numbers generator"));
00692     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
00693     options->label_threads = (GtkLabel *)
00694         gtk_label_new (_("Threads number for the stochastic algorithm"));
00695     options->spin_threads =
00696         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00697     gtk_widget_set_tooltip_text
00698         (GTK_WIDGET (options->spin_threads),
00699          _("Number of threads to perform the calibration/optimization for "
00700            "the stochastic algorithm"));
00701     gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00702     options->label_climbing = (GtkLabel *)
00703         gtk_label_new (_("Threads number for the hill climbing method"));
00704     options->spin_climbing =
00705         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00706     gtk_widget_set_tooltip_text
00707         (GTK_WIDGET (options->spin_climbing),
00708          _("Number of threads to perform the calibration/optimization for the "
00709            "hill climbing method"));
00710     gtk_spin_button_set_value (options->spin_climbing,
00711                               (gdouble) nthreads_climbing);
00712     options->grid = (GtkGrid *) gtk_grid_new ();
00713     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00714     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00715     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00716                     0, 1, 1, 1);
00717     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00718                     1, 1, 1, 1);
00719     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
00720                     1);
00721     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
00722                     1);
00723 #if !GTK4
00724     gtk_widget_show_all (GTK_WIDGET (options->grid));
00725 #else
00726     gtk_widget_show (GTK_WIDGET (options->grid));
00727 #endif
00728     options->dialog = (GtkDialog *)
00729         gtk_dialog_new_with_buttons (_("Options"),
00730                                     window->window,
00731                                     GTK_DIALOG_MODAL,
00732                                     _("_OK"), GTK_RESPONSE_OK,
00733                                     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00734     gtk_box_append (GTK_BOX (gtk_dialog_get_content_area (options->dialog)),
00735                    GTK_WIDGET (options->grid));
00736     g_signal_connect (options->dialog, "response",
00737                      G_CALLBACK (dialog_options_close), NULL);
00738     gtk_window_present (GTK_WINDOW (options->dialog));
00739 #if DEBUG_INTERFACE
00740     fprintf (stderr, "options_new: end\n");
00741 #endif
00742 }

```

Here is the call graph for this function:



#### 4.11.3.17 running\_new()

```
static void running_new ( ) [inline], [static]
```

Function to open the running dialog.

Definition at line 748 of file [interface.c](#).

```
00749 {
00750     #if DEBUG_INTERFACE
00751         fprintf (stderr, "running_new: start\n");
00752     #endif
00753     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00754     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00755     running->grid = (GtkGrid *) gtk_grid_new ();
00756     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00757     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00758     running->dialog = (GtkDialog *)
00759         gtk_dialog_new_with_buttons (_("Calculating"),
00760                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00761     gtk_window_set_child (GTK_WINDOW
00762         (gtk_dialog_get_content_area (running->dialog)),
00763         GTK_WIDGET (running->grid));
00764     gtk_spinner_start (running->spinner);
00765     #if !GTK4
00766         gtk_widget_show_all (GTK_WIDGET (running->dialog));
00767     #else
00768         gtk_widget_show (GTK_WIDGET (running->dialog));
00769     #endif
00770     #if DEBUG_INTERFACE
00771         fprintf (stderr, "running_new: end\n");
00772     #endif
00773 }
```

#### 4.11.3.18 window\_about()

```
static void window_about ( ) [static]
```

Function to show an about dialog.

Definition at line 1120 of file [interface.c](#).

```
01121 {
01122     static const gchar *authors[] = {
01123         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01124         "Borja Latorre Garcés <borja.latorre@csic.es>",
01125         NULL
01126     };
01127     #if DEBUG_INTERFACE
01128         fprintf (stderr, "window_about: start\n");
01129     #endif
01130     gtk_show_about_dialog
01131         (window->window,
01132          "program_name", "MPCOTool",
01133          "comments",
01134          _("The Multi-Purposes Calibration and Optimization Tool.\n"
01135            "A software to perform calibrations or optimizations of empirical "
01136            "parameters"),
01137          "authors", authors,
01138          "translator-credits",
01139          "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01140            "(english, french and spanish)\n"
01141            "Uğur Çayoğlu (german)",
01142          "version", "4.10.2",
01143          "copyright", "Copyright 2012-2023 Javier Burguete Tolosa",
01144          "logo", window->logo,
01145          "website", "https://github.com/jburguete/mpcotool",
01146          "license-type", GTK_LICENSE_BSD, NULL);
01147     #if DEBUG_INTERFACE
01148         fprintf (stderr, "window_about: end\n");
01149     #endif
01150 }
```

#### 4.11.3.19 window\_add\_experiment()

```
static void window_add_experiment ( ) [static]
```

Function to add an experiment in the main window.



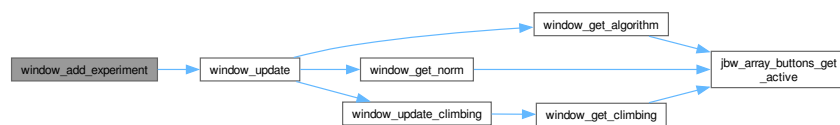
Definition at line 1438 of file [interface.c](#).

```

1439 {
1440     unsigned int i, j;
1441     #if DEBUG_INTERFACE
1442     fprintf (stderr, "window_add_experiment: start\n");
1443     #endif
1444     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
1445     g_signal_handler_block (window->combo_experiment, window->id_experiment);
1446     gtk_combo_box_text_insert_text
1447     (window->combo_experiment, i, input->experiment[i].name);
1448     g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
1449     input->experiment = (Experiment *) g_realloc
1450     (input->experiment, (input->nexperiments + 1) * sizeof (Experiment));
1451     for (j = input->nexperiments - 1; j > i; --j)
1452         memcpy (input->experiment + j + 1, input->experiment + j,
1453             sizeof (Experiment));
1454     input->experiment[j + 1].weight = input->experiment[j].weight;
1455     input->experiment[j + 1].ninputs = input->experiment[j].ninputs;
1456     if (input->type == INPUT_TYPE_XML)
1457     {
1458         input->experiment[j + 1].name
1459         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
1460         for (j = 0; j < input->experiment->ninputs; ++j)
1461             input->experiment[i + 1].stencil[j]
1462             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].stencil[j]);
1463     }
1464     else
1465     {
1466         input->experiment[j + 1].name = g_strdup (input->experiment[j].name);
1467         for (j = 0; j < input->experiment->ninputs; ++j)
1468             input->experiment[i + 1].stencil[j]
1469             = g_strdup (input->experiment[i].stencil[j]);
1470     }
1471     ++input->nexperiments;
1472     for (j = 0; j < input->experiment->ninputs; ++j)
1473         g_signal_handler_block (window->button_template[j], window->id_input[j]);
1474     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
1475     for (j = 0; j < input->experiment->ninputs; ++j)
1476         g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
1477     window_update ();
1478     #if DEBUG_INTERFACE
1479     fprintf (stderr, "window_add_experiment: end\n");
1480     #endif
1481 }

```

Here is the call graph for this function:



#### 4.11.3.20 window\_add\_variable()

```
static void window_add_variable ( ) [static]
```

Function to add a variable in the main window.

Definition at line 1775 of file [interface.c](#).

```

1776 {
1777     unsigned int i, j;
1778     #if DEBUG_INTERFACE
1779     fprintf (stderr, "window_add_variable: start\n");
1780     #endif
1781     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
1782     g_signal_handler_block (window->combo_variable, window->id_variable);
1783     gtk_combo_box_text_insert_text (window->combo_variable, i,
1784         input->variable[i].name);
1785     g_signal_handler_unblock (window->combo_variable, window->id_variable);
1786     input->variable = (Variable *) g_realloc

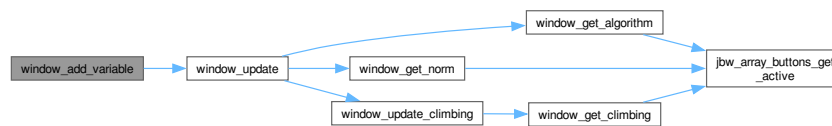
```

```

01787     (input->variable, (input->nvariables + 1) * sizeof (Variable));
01788     for (j = input->nvariables - 1; j > i; --j)
01789         memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01790     memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01791     if (input->type == INPUT_TYPE_XML)
01792         input->variable[j + 1].name
01793         = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01794     else
01795         input->variable[j + 1].name = g_strdup (input->variable[j].name);
01796     ++input->nvariables;
01797     g_signal_handler_block (window->entry_variable, window->id_variable_label);
01798     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01799     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01800     window_update ();
01801     #if DEBUG_INTERFACE
01802     fprintf (stderr, "window_add_variable: end\n");
01803     #endif
01804 }

```

Here is the call graph for this function:



#### 4.11.3.21 window\_get\_algorithm()

```
static unsigned int window_get_algorithm ( ) [static]
```

Function to get the stochastic algorithm number.

##### Returns

Stochastic algorithm number.

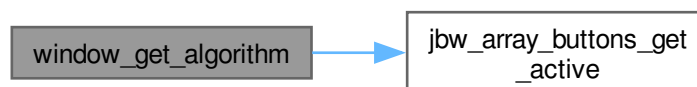
Definition at line 781 of file [interface.c](#).

```

00782 {
00783     unsigned int i;
00784     #if DEBUG_INTERFACE
00785     fprintf (stderr, "window_get_algorithm: start\n");
00786     #endif
00787     i = jbw_array_buttons_get_active (window->button_algorithm, NALGORITHMS);
00788     #if DEBUG_INTERFACE
00789     fprintf (stderr, "window_get_algorithm: %u\n", i);
00790     fprintf (stderr, "window_get_algorithm: end\n");
00791     #endif
00792     return i;
00793 }

```

Here is the call graph for this function:



#### 4.11.3.22 window\_get\_climbing()

```
static unsigned int window_get_climbing ( ) [static]
```

Function to get the hill climbing method number.

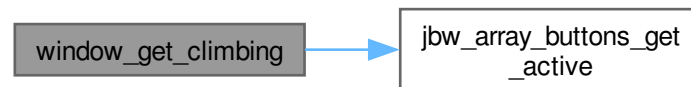
##### Returns

Hill climbing method number.

Definition at line 801 of file [interface.c](#).

```
00802 {  
00803     unsigned int i;  
00804     #if DEBUG_INTERFACE  
00805     fprintf (stderr, "window_get_climbing: start\n");  
00806     #endif  
00807     i = jbw_array_buttons_get_active (window->button_climbing, NCLIMBINGS);  
00808     #if DEBUG_INTERFACE  
00809     fprintf (stderr, "window_get_climbing: %u\n", i);  
00810     fprintf (stderr, "window_get_climbing: end\n");  
00811     #endif  
00812     return i;  
00813 }
```

Here is the call graph for this function:



#### 4.11.3.23 window\_get\_norm()

```
static unsigned int window_get_norm ( ) [static]
```

Function to get the norm method number.

##### Returns

Norm method number.

Definition at line 821 of file [interface.c](#).

```
00822 {  
00823     unsigned int i;  
00824     #if DEBUG_INTERFACE  
00825     fprintf (stderr, "window_get_norm: start\n");  
00826     #endif  
00827     i = jbw_array_buttons_get_active (window->button_norm, NNORMS);  
00828     #if DEBUG_INTERFACE  
00829     fprintf (stderr, "window_get_norm: %u\n", i);  
00830     fprintf (stderr, "window_get_norm: end\n");  
00831     #endif  
00832     return i;  
00833 }
```

Here is the call graph for this function:



#### 4.11.3.24 window\_help()

```
static void window_help ( ) [static]
```

Function to show a help dialog.

Definition at line 1092 of file [interface.c](#).

```

01093 {
01094     char *buffer, *buffer2;
01095     #if DEBUG_INTERFACE
01096     fprintf (stderr, "window_help: start\n");
01097     #endif
01098     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01099                               _("user-manual.pdf"), NULL);
01100     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01101     g_free (buffer2);
01102     #if GTK4
01103     gtk_show_uri (window->window, buffer, GDK_CURRENT_TIME);
01104     #else
01105     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01106     #endif
01107     #if DEBUG_INTERFACE
01108     fprintf (stderr, "window_help: uri=%s\n", buffer);
01109     #endif
01110     g_free (buffer);
01111     #if DEBUG_INTERFACE
01112     fprintf (stderr, "window_help: end\n");
01113     #endif
01114 }
  
```

#### 4.11.3.25 window\_inputs\_experiment()

```
static void window_inputs_experiment ( ) [static]
```

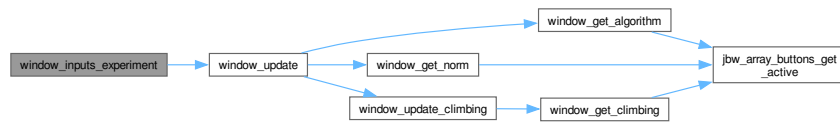
Function to update the experiment input templates number in the main window.

Definition at line 1571 of file [interface.c](#).

```

01572 {
01573     unsigned int j;
01574     #if DEBUG_INTERFACE
01575     fprintf (stderr, "window_inputs_experiment: start\n");
01576     #endif
01577     j = input->experiment->ninputs - 1;
01578     if (j && !gtk_check_button_get_active (window->check_template[j]))
01579         --input->experiment->ninputs;
01580     if (input->experiment->ninputs < MAX_NINPUTS
01581         && gtk_check_button_get_active (window->check_template[j]))
01582         ++input->experiment->ninputs;
01583     window_update ();
01584     #if DEBUG_INTERFACE
01585     fprintf (stderr, "window_inputs_experiment: end\n");
01586     #endif
01587 }
  
```

Here is the call graph for this function:



#### 4.11.3.26 window\_label\_variable()

```
static void window_label_variable ( ) [static]
```

Function to set the variable label in the main window.

Definition at line 1810 of file [interface.c](#).

```

01811 {
01812     unsigned int i;
01813     const char *buffer;
01814     #if DEBUG_INTERFACE
01815     fprintf (stderr, "window_label_variable: start\n");
01816     #endif
01817     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01818     buffer = gtk_entry_get_text (window->entry_variable);
01819     g_signal_handler_block (window->combo_variable, window->id_variable);
01820     gtk_combo_box_text_remove (window->combo_variable, i);
01821     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01822     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01823     g_signal_handler_unblock (window->combo_variable, window->id_variable);
01824     #if DEBUG_INTERFACE
01825     fprintf (stderr, "window_label_variable: end\n");
01826     #endif
01827 }
  
```

Here is the call graph for this function:



#### 4.11.3.27 window\_name\_experiment()

```
static void window_name_experiment ( ) [static]
```

Function to set the experiment name in the main window.

Definition at line 1518 of file [interface.c](#).

```

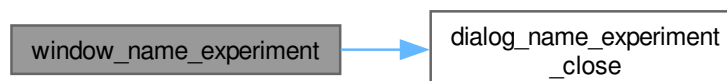
01519 {
01520     GtkFileChooserDialog *dlg;
01521     GMainLoop *loop;
01522     const char *buffer;
01523     unsigned int i;
01524     #if DEBUG_INTERFACE
  
```

```

01525     fprintf (stderr, "window_name_experiment: start\n");
01526 #endif
01527     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01528     buffer = gtk_button_get_label (window->button_experiment);
01529     dlg = (GtkFileChooserDialog *)
01530         gtk_file_chooser_dialog_new (_("Open experiment file"),
01531                                     window->window,
01532                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01533                                     _("_Cancel"),
01534                                     GTK_RESPONSE_CANCEL,
01535                                     _("_Open"), GTK_RESPONSE_OK, NULL);
01536     gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01537     g_signal_connect (dlg, "response", G_CALLBACK (dialog_name_experiment_close),
01538                     (void *) (size_t) i);
01539     gtk_window_present (GTK_WINDOW (dlg));
01540     loop = g_main_loop_new (NULL, 0);
01541     g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01542                             loop);
01543     g_main_loop_run (loop);
01544     g_main_loop_unref (loop);
01545 #if DEBUG_INTERFACE
01546     fprintf (stderr, "window_name_experiment: end\n");
01547 #endif
01548 }

```

Here is the call graph for this function:



#### 4.11.3.28 window\_new()

```

void window_new (
    GtkApplication * application )

```

Function to open the main window.

##### Parameters

<i>application</i>	GtkApplication struct.
--------------------	------------------------

Definition at line 2363 of file [interface.c](#).

```

02364 {
02365     unsigned int i;
02366     char *buffer, *buffer2, buffer3[64];
02367     const char *label_algorithm[NALGORITHMS] = {
02368         "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02369     };
02370     const char *tip_algorithm[NALGORITHMS] = {
02371         _("_Monte-Carlo brute force algorithm"),
02372         _("_Sweep brute force algorithm"),
02373         _("_Genetic algorithm"),
02374         _("_Orthogonal sampling brute force algorithm"),
02375     };
02376     const char *label_climbing[NCLIMBINGS] = {
02377         _("_Coordinates climbing"), _("_Random climbing")
02378     };
02379     const char *tip_climbing[NCLIMBINGS] = {
02380         _("_Coordinates climbing estimate method"),
02381         _("_Random climbing estimate method")

```

```

02382     };
02383     const char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02384     const char *tip_norm[NNORMS] = {
02385         _("Euclidean error norm (L2)"),
02386         _("Maximum error norm (L)"),
02387         _("P error norm (Lp)"),
02388         _("Taxicab error norm (L1)")
02389     };
02390     #if !GTK4
02391     const char *close = "delete-event";
02392     #else
02393     const char *close = "close-request";
02394     #endif
02395
02396     #if DEBUG_INTERFACE
02397     fprintf (stderr, "window_new: start\n");
02398     #endif
02399
02400     // Creating the window
02401     window->window = window_parent = main_window
02402     = (GtkWindow *) gtk_application_window_new (application);
02403
02404     // Finish when closing the window
02405     g_signal_connect_swapped (window->window, close,
02406         G_CALLBACK (g_application_quit),
02407         G_APPLICATION (application));
02408
02409     // Setting the window title
02410     gtk_window_set_title (window->window, "MPCOTool");
02411
02412     // Creating the open button
02413     window->button_open = (GtkButton *)
02414     #if !GTK4
02415     gtk_button_new_from_icon_name ("document-open", GTK_ICON_SIZE_BUTTON);
02416     #else
02417     gtk_button_new_from_icon_name ("document-open");
02418     #endif
02419     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_open),
02420         _("Open a case"));
02421     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02422
02423     // Creating the save button
02424     window->button_save = (GtkButton *)
02425     #if !GTK4
02426     gtk_button_new_from_icon_name ("document-save", GTK_ICON_SIZE_BUTTON);
02427     #else
02428     gtk_button_new_from_icon_name ("document-save");
02429     #endif
02430     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_save),
02431         _("Save the case"));
02432     g_signal_connect (window->button_save, "clicked", (GCallback) window_save,
02433         NULL);
02434
02435     // Creating the run button
02436     window->button_run = (GtkButton *)
02437     #if !GTK4
02438     gtk_button_new_from_icon_name ("system-run", GTK_ICON_SIZE_BUTTON);
02439     #else
02440     gtk_button_new_from_icon_name ("system-run");
02441     #endif
02442     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_run),
02443         _("Run the optimization"));
02444     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02445
02446     // Creating the options button
02447     window->button_options = (GtkButton *)
02448     #if !GTK4
02449     gtk_button_new_from_icon_name ("preferences-system", GTK_ICON_SIZE_BUTTON);
02450     #else
02451     gtk_button_new_from_icon_name ("preferences-system");
02452     #endif
02453     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_options),
02454         _("Edit the case"));
02455     g_signal_connect (window->button_options, "clicked", options_new, NULL);
02456
02457     // Creating the help button
02458     window->button_help = (GtkButton *)
02459     #if !GTK4
02460     gtk_button_new_from_icon_name ("help-browser", GTK_ICON_SIZE_BUTTON);
02461     #else
02462     gtk_button_new_from_icon_name ("help-browser");
02463     #endif
02464     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_help), _("Help"));
02465     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02466
02467     // Creating the about button
02468     window->button_about = (GtkButton *)

```

```

02469 #if !GTK4
02470     gtk_button_new_from_icon_name ("help-about", GTK_ICON_SIZE_BUTTON);
02471 #else
02472     gtk_button_new_from_icon_name ("help-about");
02473 #endif
02474     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_about), _("About"));
02475     g_signal_connect (window->button_about, "clicked", window_about, NULL);
02476
02477     // Creating the exit button
02478     window->button_exit = (GtkButton *)
02479 #if !GTK4
02480     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
02481 #else
02482     gtk_button_new_from_icon_name ("application-exit");
02483 #endif
02484     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_exit), _("Exit"));
02485     g_signal_connect_swapped (window->button_exit, "clicked",
02486                             G_CALLBACK (g_application_quit),
02487                             G_APPLICATION (application));
02488
02489     // Creating the buttons bar
02490     window->box_buttons = (GtkBox *) gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
02491     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_open));
02492     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_save));
02493     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_run));
02494     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_options));
02495     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_help));
02496     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_about));
02497     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_exit));
02498
02499     // Creating the simulator program label and entry
02500     window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02501     window->button_simulator = (GtkButton *)
02502     gtk_button_new_with_mnemonic (_("Simulator program"));
02503     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02504                                 _("Simulator program executable file"));
02505     gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02506     g_signal_connect (window->button_simulator, "clicked",
02507                     G_CALLBACK (dialog_simulator), NULL);
02508
02509     // Creating the evaluator program label and entry
02510     window->check_evaluator = (GtkCheckButton *)
02511     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02512     g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
02513     window->button_evaluator = (GtkButton *)
02514     gtk_button_new_with_mnemonic (_("Evaluator program"));
02515     gtk_widget_set_tooltip_text
02516     (GTK_WIDGET (window->button_evaluator),
02517      _("Optional evaluator program executable file"));
02518     g_signal_connect (window->button_evaluator, "clicked",
02519                     G_CALLBACK (dialog_evaluator), NULL);
02520
02521     // Creating the cleaner program label and entry
02522     window->check_cleaner = (GtkCheckButton *)
02523     gtk_check_button_new_with_mnemonic (_("_Cleaner program"));
02524     g_signal_connect (window->check_cleaner, "toggled", window_update, NULL);
02525     window->button_cleaner = (GtkButton *)
02526     gtk_button_new_with_mnemonic (_("Cleaner program"));
02527     gtk_widget_set_tooltip_text
02528     (GTK_WIDGET (window->button_cleaner),
02529      _("Optional cleaner program executable file"));
02530     g_signal_connect (window->button_cleaner, "clicked",
02531                     G_CALLBACK (dialog_cleaner), NULL);
02532
02533     // Creating the results files labels and entries
02534     window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02535     window->entry_result = (GtkEntry *) gtk_entry_new ();
02536     gtk_widget_set_tooltip_text
02537     (GTK_WIDGET (window->entry_result), _("Best results file"));
02538     window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02539     window->entry_variables = (GtkEntry *) gtk_entry_new ();
02540     gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02542
02543     // Creating the files grid and attaching widgets
02544     window->grid_files = (GtkGrid *) gtk_grid_new ();
02545     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
02546                     0, 0, 1, 1);
02547     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
02548                     1, 0, 1, 1);
02549     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
02550                     0, 1, 1, 1);
02551     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
02552                     1, 1, 1, 1);
02553     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_cleaner),
02554                     0, 2, 1, 1);
02555     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_cleaner),

```



```

02556         1, 2, 1, 1);
02557     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
02558                     0, 3, 1, 1);
02559     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
02560                     1, 3, 1, 1);
02561     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
02562                     0, 4, 1, 1);
02563     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
02564                     1, 4, 1, 1);
02565
02566     // Creating the algorithm properties
02567     window->label_simulations = (GtkLabel *) gtk_label_new
02568         (_("Simulations number"));
02569     window->spin_simulations
02570         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02571     gtk_widget_set_tooltip_text
02572         (GTK_WIDGET (window->spin_simulations),
02573          _("Number of simulations to perform for each iteration"));
02574     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02575     window->label_iterations = (GtkLabel *)
02576         gtk_label_new (_("Iterations number"));
02577     window->spin_iterations
02578         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02579     gtk_widget_set_tooltip_text
02580         (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02581     g_signal_connect
02582         (window->spin_iterations, "value-changed", window_update, NULL);
02583     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02584     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02585     window->spin_tolerance =
02586         (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02587     gtk_widget_set_tooltip_text
02588         (GTK_WIDGET (window->spin_tolerance),
02589          _("Tolerance to set the variable interval on the next iteration"));
02590     window->label_best = (GtkLabel *) gtk_label_new (_("Bests number"));
02591     window->spin_best
02592         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02593     gtk_widget_set_tooltip_text
02594         (GTK_WIDGET (window->spin_best),
02595          _("Number of best simulations used to set the variable interval "
02596            "on the next iteration"));
02597     window->label_population
02598         = (GtkLabel *) gtk_label_new (_("Population number"));
02599     window->spin_population
02600         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02601     gtk_widget_set_tooltip_text
02602         (GTK_WIDGET (window->spin_population),
02603          _("Number of population for the genetic algorithm"));
02604     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02605     window->label_generations
02606         = (GtkLabel *) gtk_label_new (_("Generations number"));
02607     window->spin_generations
02608         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02609     gtk_widget_set_tooltip_text
02610         (GTK_WIDGET (window->spin_generations),
02611          _("Number of generations for the genetic algorithm"));
02612     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02613     window->spin_mutation
02614         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02615     gtk_widget_set_tooltip_text
02616         (GTK_WIDGET (window->spin_mutation),
02617          _("Ratio of mutation for the genetic algorithm"));
02618     window->label_reproduction
02619         = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02620     window->spin_reproduction
02621         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02622     gtk_widget_set_tooltip_text
02623         (GTK_WIDGET (window->spin_reproduction),
02624          _("Ratio of reproduction for the genetic algorithm"));
02625     window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02626     window->spin_adaptation
02627         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02628     gtk_widget_set_tooltip_text
02629         (GTK_WIDGET (window->spin_adaptation),
02630          _("Ratio of adaptation for the genetic algorithm"));
02631     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02632     window->spin_threshold = (GtkSpinButton *)
02633         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02634                                         precision[DEFAULT_PRECISION]);
02635     gtk_widget_set_tooltip_text
02636         (GTK_WIDGET (window->spin_threshold),
02637          _("Threshold in the objective function to finish the simulations"));
02638     window->scrolled_threshold = (GtkScrolledWindow *)
02639         #if !GTK4
02640         gtk_scrolled_window_new (NULL, NULL);
02641     #else
02642         gtk_scrolled_window_new ();

```

```

02643 #endif
02644 gtk_scrolled_window_set_child (window->scrolled_threshold,
02645                                GTK_WIDGET (window->spin_threshold));
02646 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02647 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02648 //                          GTK_ALIGN_FILL);
02649 // Creating the hill climbing method properties
02650 window->check_climbing = (GtkCheckButton *)
02651     gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02652 g_signal_connect (window->check_climbing, "toggled", window_update, NULL);
02653 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02654 #if !GTK4
02655 window->button_climbing[0] = (GtkRadioButton *)
02656     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02657 #else
02658 window->button_climbing[0] = (GtkCheckButton *)
02659     gtk_check_button_new_with_mnemonic (label_climbing[0]);
02660 #endif
02661 gtk_grid_attach (window->grid_climbing,
02662                 GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02663 g_signal_connect (window->button_climbing[0], "toggled", window_update, NULL);
02664 for (i = 0; ++i < NCLIMBINGS;)
02665 {
02666 #if !GTK4
02667 window->button_climbing[i] = (GtkRadioButton *)
02668     gtk_radio_button_new_with_mnemonic
02669     (gtk_radio_button_get_group (window->button_climbing[0]),
02670      label_climbing[i]);
02671 #else
02672 window->button_climbing[i] = (GtkCheckButton *)
02673     gtk_check_button_new_with_mnemonic (label_climbing[i]);
02674 gtk_check_button_set_group (window->button_climbing[i],
02675                             window->button_climbing[0]);
02676 #endif
02677 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02678                             tip_climbing[i]);
02679 gtk_grid_attach (window->grid_climbing,
02680                 GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02681 g_signal_connect (window->button_climbing[i], "toggled", window_update,
02682                 NULL);
02683 }
02684 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02685 window->spin_steps = (GtkSpinButton *)
02686     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02687 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02688 window->label_final_steps
02689     = (GtkLabel *) gtk_label_new (_("Final steps number"));
02690 window->spin_final_steps = (GtkSpinButton *)
02691     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02692 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_final_steps), TRUE);
02693 window->label_estimates
02694     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02695 window->spin_estimates = (GtkSpinButton *)
02696     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02697 window->label_relaxation
02698     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02699 window->spin_relaxation = (GtkSpinButton *)
02700     gtk_spin_button_new_with_range (0., 2., 0.001);
02701 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
02702                 0, NCLIMBINGS, 1, 1);
02703 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
02704                 1, NCLIMBINGS, 1, 1);
02705 gtk_grid_attach (window->grid_climbing,
02706                 GTK_WIDGET (window->label_final_steps),
02707                 0, NCLIMBINGS + 1, 1, 1);
02708 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_final_steps),
02709                 1, NCLIMBINGS + 1, 1, 1);
02710 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
02711                 0, NCLIMBINGS + 2, 1, 1);
02712 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
02713                 1, NCLIMBINGS + 2, 1, 1);
02714 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
02715                 0, NCLIMBINGS + 3, 1, 1);
02716 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
02717                 1, NCLIMBINGS + 3, 1, 1);
02718 // Creating the array of algorithms
02719 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02720 #if !GTK4
02721 window->button_algorithm[0] = (GtkRadioButton *)
02722     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02723 #else
02724 window->button_algorithm[0] = (GtkCheckButton *)
02725     gtk_check_button_new_with_mnemonic (label_algorithm[0]);
02726 #endif
02727 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),

```

```

02730             tip_algorithm[0]);
02731     gtk_grid_attach (window->grid_algorithm,
02732                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02733     g_signal_connect (window->button_algorithm[0], "toggled",
02734                       window_set_algorithm, NULL);
02735     for (i = 0; ++i < NALGORITHMS;)
02736     {
02737 #if !GTK4
02738         window->button_algorithm[i] = (GtkRadioButton *)
02739             gtk_radio_button_new_with_mnemonic
02740             (gtk_radio_button_get_group (window->button_algorithm[0]),
02741              label_algorithm[i]);
02742 #else
02743         window->button_algorithm[i] = (GtkCheckButton *)
02744             gtk_check_button_new_with_mnemonic (label_algorithm[i]);
02745         gtk_check_button_set_group (window->button_algorithm[i],
02746                                    window->button_algorithm[0]);
02747 #endif
02748         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02749                                     tip_algorithm[i]);
02750         gtk_grid_attach (window->grid_algorithm,
02751                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02752         g_signal_connect (window->button_algorithm[i], "toggled",
02753                           window_set_algorithm, NULL);
02754     }
02755     gtk_grid_attach (window->grid_algorithm,
02756                     GTK_WIDGET (window->label_simulations),
02757                     0, NALGORITHMS, 1, 1);
02758     gtk_grid_attach (window->grid_algorithm,
02759                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02760     gtk_grid_attach (window->grid_algorithm,
02761                     GTK_WIDGET (window->label_iterations),
02762                     0, NALGORITHMS + 1, 1, 1);
02763     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
02764                     1, NALGORITHMS + 1, 1, 1);
02765     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
02766                     0, NALGORITHMS + 2, 1, 1);
02767     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
02768                     1, NALGORITHMS + 2, 1, 1);
02769     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
02770                     0, NALGORITHMS + 3, 1, 1);
02771     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
02772                     1, NALGORITHMS + 3, 1, 1);
02773     gtk_grid_attach (window->grid_algorithm,
02774                     GTK_WIDGET (window->label_population),
02775                     0, NALGORITHMS + 4, 1, 1);
02776     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
02777                     1, NALGORITHMS + 4, 1, 1);
02778     gtk_grid_attach (window->grid_algorithm,
02779                     GTK_WIDGET (window->label_generations),
02780                     0, NALGORITHMS + 5, 1, 1);
02781     gtk_grid_attach (window->grid_algorithm,
02782                     GTK_WIDGET (window->spin_generations),
02783                     1, NALGORITHMS + 5, 1, 1);
02784     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
02785                     0, NALGORITHMS + 6, 1, 1);
02786     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
02787                     1, NALGORITHMS + 6, 1, 1);
02788     gtk_grid_attach (window->grid_algorithm,
02789                     GTK_WIDGET (window->label_reproduction),
02790                     0, NALGORITHMS + 7, 1, 1);
02791     gtk_grid_attach (window->grid_algorithm,
02792                     GTK_WIDGET (window->spin_reproduction),
02793                     1, NALGORITHMS + 7, 1, 1);
02794     gtk_grid_attach (window->grid_algorithm,
02795                     GTK_WIDGET (window->label_adaptation),
02796                     0, NALGORITHMS + 8, 1, 1);
02797     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),
02798                     1, NALGORITHMS + 8, 1, 1);
02799     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
02800                     0, NALGORITHMS + 9, 2, 1);
02801     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
02802                     0, NALGORITHMS + 10, 2, 1);
02803     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
02804                     0, NALGORITHMS + 11, 1, 1);
02805     gtk_grid_attach (window->grid_algorithm,
02806                     GTK_WIDGET (window->scrolled_threshold),
02807                     1, NALGORITHMS + 11, 1, 1);
02808     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02809     gtk_frame_set_child (window->frame_algorithm,
02810                         GTK_WIDGET (window->grid_algorithm));
02811
02812     // Creating the variable widgets
02813     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02814     gtk_widget_set_tooltip_text
02815         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02816     window->id_variable = g_signal_connect

```

```

02817     (window->combo_variable, "changed", window_set_variable, NULL);
02818 #if !GTK4
02819     window->button_add_variable = (GtkButton *)
02820     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02821 #else
02822     window->button_add_variable = (GtkButton *)
02823     gtk_button_new_from_icon_name ("list-add");
02824 #endif
02825     g_signal_connect (window->button_add_variable, "clicked", window_add_variable,
02826     NULL);
02827     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02828     _("Add variable"));
02829 #if !GTK4
02830     window->button_remove_variable = (GtkButton *)
02831     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02832 #else
02833     window->button_remove_variable = (GtkButton *)
02834     gtk_button_new_from_icon_name ("list-remove");
02835 #endif
02836     g_signal_connect (window->button_remove_variable, "clicked",
02837     window_remove_variable, NULL);
02838     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02839     _("Remove variable"));
02840     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02841     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02842     gtk_widget_set_tooltip_text
02843     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02844     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02845     window->id_variable_label = g_signal_connect
02846     (window->entry_variable, "changed", window_label_variable, NULL);
02847     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02848     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02849     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02850     gtk_widget_set_tooltip_text
02851     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02852     window->scrolled_min = (GtkScrolledWindow *)
02853     #if !GTK4
02854     gtk_scrolled_window_new (NULL, NULL);
02855     #else
02856     gtk_scrolled_window_new ();
02857     #endif
02858     gtk_scrolled_window_set_child (window->scrolled_min,
02859     GTK_WIDGET (window->spin_min));
02860     g_signal_connect (window->spin_min, "value-changed",
02861     window_rangemin_variable, NULL);
02862     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02863     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02864     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02865     gtk_widget_set_tooltip_text
02866     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02867     window->scrolled_max = (GtkScrolledWindow *)
02868     #if !GTK4
02869     gtk_scrolled_window_new (NULL, NULL);
02870     #else
02871     gtk_scrolled_window_new ();
02872     #endif
02873     gtk_scrolled_window_set_child (window->scrolled_max,
02874     GTK_WIDGET (window->spin_max));
02875     g_signal_connect (window->spin_max, "value-changed",
02876     window_rangemax_variable, NULL);
02877     window->check_minabs = (GtkCheckButton *)
02878     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02879     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02880     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02881     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02882     gtk_widget_set_tooltip_text
02883     (GTK_WIDGET (window->spin_minabs),
02884     _("Minimum allowed value of the variable"));
02885     window->scrolled_minabs = (GtkScrolledWindow *)
02886     #if !GTK4
02887     gtk_scrolled_window_new (NULL, NULL);
02888     #else
02889     gtk_scrolled_window_new ();
02890     #endif
02891     gtk_scrolled_window_set_child (window->scrolled_minabs,
02892     GTK_WIDGET (window->spin_minabs));
02893     g_signal_connect (window->spin_minabs, "value-changed",
02894     window_rangeminabs_variable, NULL);
02895     window->check_maxabs = (GtkCheckButton *)
02896     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02897     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02898     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02899     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02900     gtk_widget_set_tooltip_text
02901     (GTK_WIDGET (window->spin_maxabs),
02902     _("Maximum allowed value of the variable"));
02903     window->scrolled_maxabs = (GtkScrolledWindow *)

```

```

02904 #if !GTK4
02905     gtk_scrolled_window_new (NULL, NULL);
02906 #else
02907     gtk_scrolled_window_new ();
02908 #endif
02909     gtk_scrolled_window_set_child (window->scrolled_maxabs,
02910                                     GTK_WIDGET (window->spin_maxabs));
02911     g_signal_connect (window->spin_maxabs, "value-changed",
02912                       window_rangemaxabs_variable, NULL);
02913     window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02914     window->spin_precision = (GtkSpinButton *)
02915         gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02916     gtk_widget_set_tooltip_text
02917         (GTK_WIDGET (window->spin_precision),
02918          _("Number of precision floating point digits\n"
02919            "0 is for integer numbers"));
02920     g_signal_connect (window->spin_precision, "value-changed",
02921                       window_precision_variable, NULL);
02922     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02923     window->spin_sweeps =
02924         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02925     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02926                                 _("Number of steps sweeping the variable"));
02927     g_signal_connect (window->spin_sweeps, "value-changed",
02928                       window_update_variable, NULL);
02929     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02930     window->spin_bits
02931         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02932     gtk_widget_set_tooltip_text
02933         (GTK_WIDGET (window->spin_bits),
02934          _("Number of bits to encode the variable"));
02935     g_signal_connect
02936         (window->spin_bits, "value-changed", window_update_variable, NULL);
02937     window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02938     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02939         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02940     gtk_widget_set_tooltip_text
02941         (GTK_WIDGET (window->spin_step),
02942          _("Initial step size for the hill climbing method"));
02943     window->scrolled_step = (GtkScrolledWindow *)
02944 #if !GTK4
02945         gtk_scrolled_window_new (NULL, NULL);
02946 #else
02947         gtk_scrolled_window_new ();
02948 #endif
02949     gtk_scrolled_window_set_child (window->scrolled_step,
02950                                     GTK_WIDGET (window->spin_step));
02951     g_signal_connect
02952         (window->spin_step, "value-changed", window_step_variable, NULL);
02953     window->grid_variable = (GtkGrid *) gtk_grid_new ();
02954     gtk_grid_attach (window->grid_variable,
02955                     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02956     gtk_grid_attach (window->grid_variable,
02957                     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02958     gtk_grid_attach (window->grid_variable,
02959                     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02960     gtk_grid_attach (window->grid_variable,
02961                     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02962     gtk_grid_attach (window->grid_variable,
02963                     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02964     gtk_grid_attach (window->grid_variable,
02965                     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02966     gtk_grid_attach (window->grid_variable,
02967                     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02968     gtk_grid_attach (window->grid_variable,
02969                     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02970     gtk_grid_attach (window->grid_variable,
02971                     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02972     gtk_grid_attach (window->grid_variable,
02973                     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02974     gtk_grid_attach (window->grid_variable,
02975                     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02976     gtk_grid_attach (window->grid_variable,
02977                     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02978     gtk_grid_attach (window->grid_variable,
02979                     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02980     gtk_grid_attach (window->grid_variable,
02981                     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02982     gtk_grid_attach (window->grid_variable,
02983                     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02984     gtk_grid_attach (window->grid_variable,
02985                     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02986     gtk_grid_attach (window->grid_variable,
02987                     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02988     gtk_grid_attach (window->grid_variable,
02989                     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02990     gtk_grid_attach (window->grid_variable,

```

```

02991         GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02992     gtk_grid_attach (window->grid_variable,
02993         GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02994     gtk_grid_attach (window->grid_variable,
02995         GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02996     window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02997     gtk_frame_set_child (window->frame_variable,
02998         GTK_WIDGET (window->grid_variable));
02999
03000     // Creating the experiment widgets
03001     window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03002     gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03003         _("Experiment selector"));
03004     window->id_experiment = g_signal_connect
03005         (window->combo_experiment, "changed", window_set_experiment, NULL);
03006     #if !GTK4
03007     window->button_add_experiment = (GtkButton *)
03008         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
03009     #else
03010     window->button_add_experiment = (GtkButton *)
03011         gtk_button_new_from_icon_name ("list-add");
03012     #endif
03013     g_signal_connect
03014         (window->button_add_experiment, "clicked", window_add_experiment, NULL);
03015     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03016         _("Add experiment"));
03017     #if !GTK4
03018     window->button_remove_experiment = (GtkButton *)
03019         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
03020     #else
03021     window->button_remove_experiment = (GtkButton *)
03022         gtk_button_new_from_icon_name ("list-remove");
03023     #endif
03024     g_signal_connect (window->button_remove_experiment, "clicked",
03025         window_remove_experiment, NULL);
03026     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03027         _("Remove experiment"));
03028     window->label_experiment
03029         = (GtkLabel *) gtk_label_new (_("Experimental data file"));
03030     window->button_experiment = (GtkButton *)
03031         gtk_button_new_with_mnemonic (_("Experimental data file"));
03032     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03033         _("Experimental data file"));
03034     g_signal_connect (window->button_experiment, "clicked",
03035         window_name_experiment, NULL);
03036     gtk_widget_set_hexexpand (GTK_WIDGET (window->button_experiment), TRUE);
03037     window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
03038     window->spin_weight
03039         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03040     gtk_widget_set_tooltip_text
03041         (GTK_WIDGET (window->spin_weight),
03042         _("Weight factor to build the objective function"));
03043     g_signal_connect
03044         (window->spin_weight, "value-changed", window_weight_experiment, NULL);
03045     window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03046     gtk_grid_attach (window->grid_experiment,
03047         GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03048     gtk_grid_attach (window->grid_experiment,
03049         GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03050     gtk_grid_attach (window->grid_experiment,
03051         GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03052     gtk_grid_attach (window->grid_experiment,
03053         GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03054     gtk_grid_attach (window->grid_experiment,
03055         GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03056     gtk_grid_attach (window->grid_experiment,
03057         GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03058     gtk_grid_attach (window->grid_experiment,
03059         GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03060     for (i = 0; i < MAX_NINPUTS; ++i)
03061     {
03062         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
03063         window->check_template[i] = (GtkCheckButton *)
03064             gtk_check_button_new_with_label (buffer3);
03065         window->id_template[i]
03066             = g_signal_connect (window->check_template[i], "toggled",
03067                 window_inputs_experiment, NULL);
03068         gtk_grid_attach (window->grid_experiment,
03069             GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03070         window->button_template[i] = (GtkButton *)
03071             gtk_button_new_with_mnemonic (_("Input template"));
03072         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
03073             _("Experimental input template file"));
03074         window->id_input[i] =
03075             g_signal_connect_swapped (window->button_template[i], "clicked",
03076                 (GCallback) window_template_experiment,

```



```

03078             (void *) (size_t) i);
03079     gtk_grid_attach (window->grid_experiment,
03080                     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03081 }
03082 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
03083 gtk_frame_set_child (window->frame_experiment,
03084                     GTK_WIDGET (window->grid_experiment));
03085
03086 // Creating the error norm widgets
03087 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
03088 window->grid_norm = (GtkGrid *) gtk_grid_new ();
03089 gtk_frame_set_child (window->frame_norm, GTK_WIDGET (window->grid_norm));
03090 #if !GTK4
03091     window->button_norm[0] = (GtkRadioButton *)
03092         gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
03093 #else
03094     window->button_norm[0] = (GtkCheckButton *)
03095         gtk_check_button_new_with_mnemonic (label_norm[0]);
03096 #endif
03097 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
03098                             tip_norm[0]);
03099 gtk_grid_attach (window->grid_norm,
03100                 GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
03101 g_signal_connect (window->button_norm[0], "toggled", window_update, NULL);
03102 for (i = 0; ++i < NNORMS;)
03103 {
03104     #if !GTK4
03105         window->button_norm[i] = (GtkRadioButton *)
03106             gtk_radio_button_new_with_mnemonic
03107                 (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
03108     #else
03109         window->button_norm[i] = (GtkCheckButton *)
03110             gtk_check_button_new_with_mnemonic (label_norm[i]);
03111         gtk_check_button_set_group (window->button_norm[i],
03112                                   window->button_norm[0]);
03113     #endif
03114     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
03115                                 tip_norm[i]);
03116     gtk_grid_attach (window->grid_norm,
03117                     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
03118     g_signal_connect (window->button_norm[i], "toggled", window_update, NULL);
03119 }
03120 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
03121 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
03122 window->spin_p = (GtkSpinButton *)
03123     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
03124 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
03125                             _("P parameter for the P error norm"));
03126 window->scrolled_p = (GtkScrolledWindow *)
03127     #if !GTK4
03128         gtk_scrolled_window_new (NULL, NULL);
03129     #else
03130         gtk_scrolled_window_new ();
03131     #endif
03132     gtk_scrolled_window_set_child (window->scrolled_p,
03133                                   GTK_WIDGET (window->spin_p));
03134     gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
03135     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
03136     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
03137                     1, 2, 1, 2);
03138
03139 // Creating the grid and attaching the widgets to the grid
03140 window->grid = (GtkGrid *) gtk_grid_new ();
03141 gtk_grid_attach (window->grid, GTK_WIDGET (window->box_buttons), 0, 0, 3, 1);
03142 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
03143 gtk_grid_attach (window->grid,
03144                 GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
03145 gtk_grid_attach (window->grid,
03146                 GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
03147 gtk_grid_attach (window->grid,
03148                 GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
03149 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
03150 gtk_window_set_child (window->window, GTK_WIDGET (window->grid));
03151
03152 // Setting the window logo
03153 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03154 #if !GTK4
03155     gtk_window_set_icon (window->window, window->logo);
03156 #endif
03157
03158 // Showing the window
03159 #if !GTK4
03160     gtk_widget_show_all (GTK_WIDGET (window->window));
03161 #else
03162     gtk_widget_show (GTK_WIDGET (window->window));
03163 #endif
03164

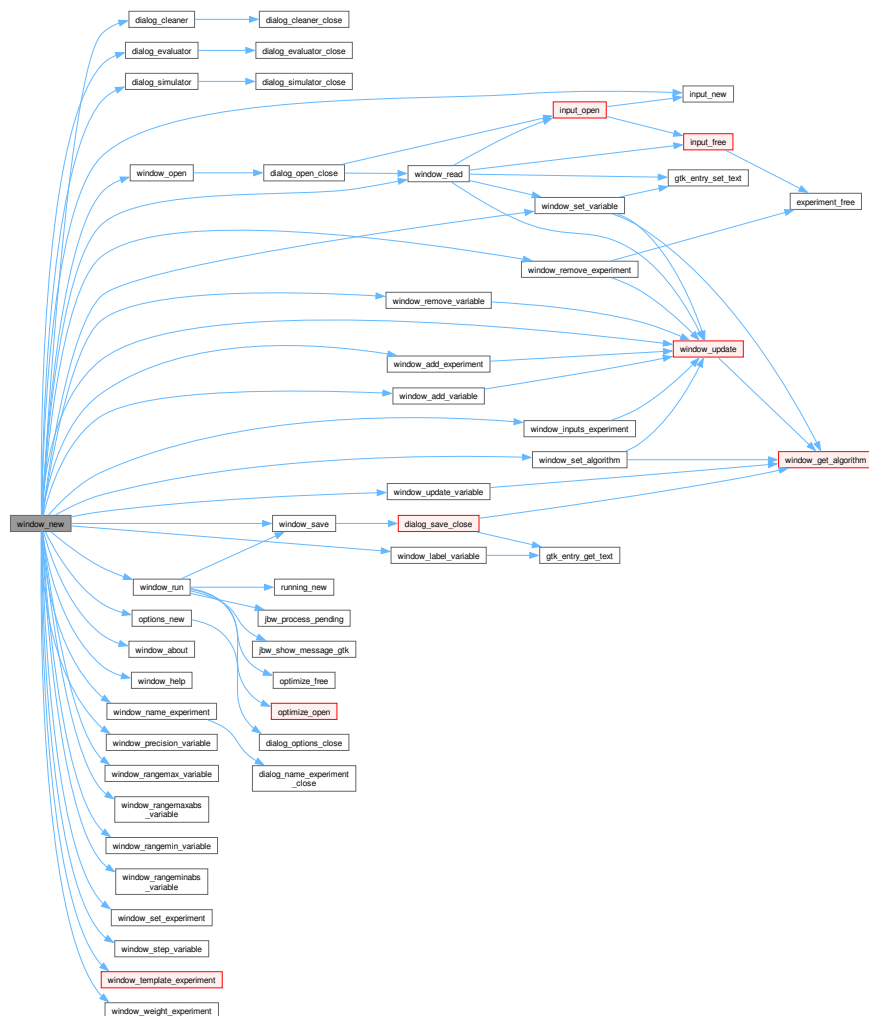
```

```

03165 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
03166 #if GTK_MINOR_VERSION >= 16
03167 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03168 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03169 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03170 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03171 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
03172 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
03173 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
03174 #endif
03175
03176 // Reading initial example
03177 input_new ();
03178 buffer2 = g_get_current_dir ();
03179 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03180 g_free (buffer2);
03181 window_read (buffer);
03182 g_free (buffer);
03183
03184 #if DEBUG_INTERFACE
03185 fprintf (stderr, "window_new: start\n");
03186 #endif
03187 }

```

Here is the call graph for this function:



#### 4.11.3.29 window\_open()

```
static void window_open ( ) [static]
```



Definition at line 2153 of file interface.c.

Here is the call graph for this function:



Generated by Doxygen

Function to update the variable precision in the main window.

Definition at line 1833 of file [interface.c](#).

```

01834 {
01835     unsigned int i;
01836     #if DEBUG_INTERFACE
01837         fprintf (stderr, "window_precision_variable: start\n");
01838     #endif
01839     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01840     input->variable[i].precision
01841     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01842     gtk_spin_button_set_digits (window->spin_min, input->variable[i].precision);
01843     gtk_spin_button_set_digits (window->spin_max, input->variable[i].precision);
01844     gtk_spin_button_set_digits (window->spin_minabs,
01845                                input->variable[i].precision);
01846     gtk_spin_button_set_digits (window->spin_maxabs,
01847                                input->variable[i].precision);
01848     #if DEBUG_INTERFACE
01849         fprintf (stderr, "window_precision_variable: end\n");
01850     #endif
01851 }
```

#### 4.11.3.31 window\_rangemax\_variable()

```
static void window_rangemax_variable ( ) [static]
```

Function to update the variable rangemax in the main window.

Definition at line 1874 of file [interface.c](#).

```

01875 {
01876     unsigned int i;
01877     #if DEBUG_INTERFACE
01878         fprintf (stderr, "window_rangemax_variable: start\n");
01879     #endif
01880     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01881     input->variable[i].rangemax = gtk_spin_button_get_value (window->spin_max);
01882     #if DEBUG_INTERFACE
01883         fprintf (stderr, "window_rangemax_variable: end\n");
01884     #endif
01885 }
```

#### 4.11.3.32 window\_rangemaxabs\_variable()

```
static void window_rangemaxabs_variable ( ) [static]
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1909 of file [interface.c](#).

```

01910 {
01911     unsigned int i;
01912     #if DEBUG_INTERFACE
01913         fprintf (stderr, "window_rangemaxabs_variable: start\n");
01914     #endif
01915     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01916     input->variable[i].rangemaxabs
01917     = gtk_spin_button_get_value (window->spin_maxabs);
01918     #if DEBUG_INTERFACE
01919         fprintf (stderr, "window_rangemaxabs_variable: end\n");
01920     #endif
01921 }
```

#### 4.11.3.33 window\_rangemin\_variable()

```
static void window_rangemin_variable ( ) [static]
```

Function to update the variable rangemin in the main window.

Definition at line 1857 of file [interface.c](#).

```

01858 {
01859     unsigned int i;
01860     #if DEBUG_INTERFACE
01861         fprintf (stderr, "window_rangemin_variable: start\n");
01862     #endif
01863     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01864     input->variable[i].rangemin = gtk_spin_button_get_value (window->spin_min);
01865     #if DEBUG_INTERFACE
01866         fprintf (stderr, "window_rangemin_variable: end\n");
01867     #endif
01868 }
```

#### 4.11.334 window\_rangeminabs\_variable()

```
static void window_rangeminabs_variable ( ) [static]
```

Function to update the variable rangeminabs in the main window.

Definition at line 1891 of file [interface.c](#).

```
01892 {
01893     unsigned int i;
01894     #if DEBUG_INTERFACE
01895     fprintf (stderr, "window_rangeminabs_variable: start\n");
01896     #endif
01897     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01898     input->variable[i].rangeminabs
01899     = gtk_spin_button_get_value (window->spin_minabs);
01900     #if DEBUG_INTERFACE
01901     fprintf (stderr, "window_rangeminabs_variable: end\n");
01902     #endif
01903 }
```

#### 4.11.335 window\_read()

```
static int window_read (
    char * filename ) [static]
```

Function to read the input data of a file.

##### Returns

1 on succes, 0 on error.

##### Parameters

<i>filename</i>	File name.
-----------------	------------

Definition at line 1983 of file [interface.c](#).

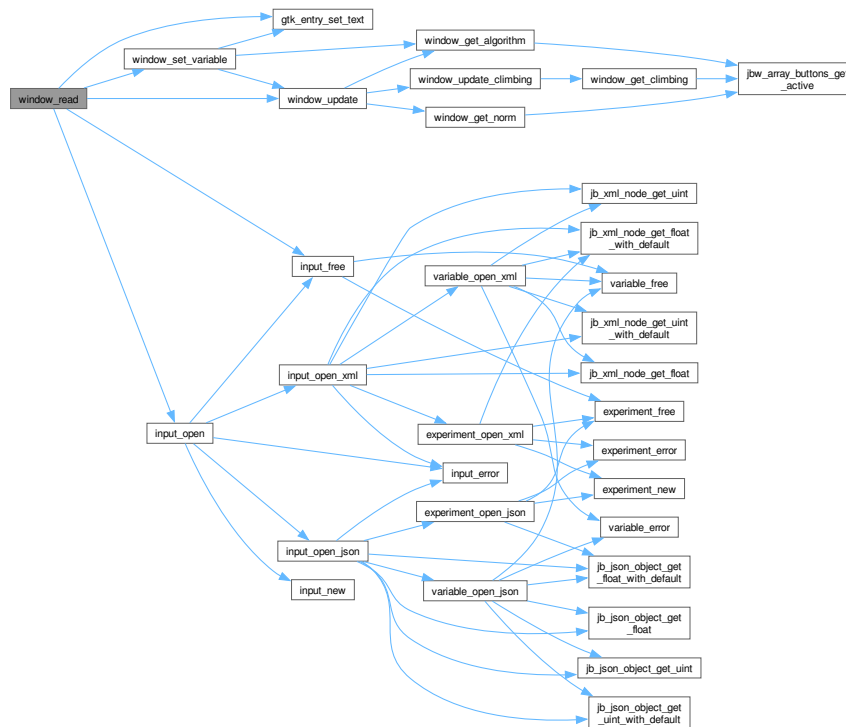
```
01984 {
01985     unsigned int i;
01986     #if DEBUG_INTERFACE
01987     fprintf (stderr, "window_read: start\n");
01988     fprintf (stderr, "window_read: file name=%s\n", filename);
01989     #endif
01990
01991     // Reading new input file
01992     input_free ();
01993     input->result = input->variables = NULL;
01994     if (!input_open (filename))
01995     {
01996     #if DEBUG_INTERFACE
01997         fprintf (stderr, "window_read: end\n");
01998     #endif
01999         return 0;
02000     }
02001
02002     // Setting GTK+ widgets data
02003     gtk_entry_set_text (window->entry_result, input->result);
02004     gtk_entry_set_text (window->entry_variables, input->variables);
02005     gtk_button_set_label (window->button_simulator, input->simulator);
02006     gtk_check_button_set_active (window->check_evaluator,
02007                                 (size_t) input->evaluator);
02008     if (input->evaluator)
02009         gtk_button_set_label (window->button_evaluator, input->evaluator);
02010     gtk_check_button_set_active (window->check_cleaner, (size_t) input->cleaner);
02011     if (input->cleaner)
02012         gtk_button_set_label (window->button_cleaner, input->cleaner);
02013     gtk_check_button_set_active (window->button_algorithm[input->algorithm],
02014                                 TRUE);
02015     switch (input->algorithm)
```

```

02016     {
02017     case ALGORITHM_MONTE_CARLO:
02018         gtk_spin_button_set_value (window->spin_simulations,
02019                                   (gdouble) input->nsimulations);
02020         // fallthrough
02021     case ALGORITHM_SWEEP:
02022     case ALGORITHM_ORTHOGONAL:
02023         gtk_spin_button_set_value (window->spin_iterations,
02024                                   (gdouble) input->niterations);
02025         gtk_spin_button_set_value (window->spin_best, (gdouble) input->nbest);
02026         gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
02027         gtk_check_button_set_active (window->check_climbing, input->nsteps);
02028         if (input->nsteps)
02029         {
02030             gtk_check_button_set_active
02031             (window->button_climbing[input->climbing], TRUE);
02032             gtk_spin_button_set_value (window->spin_steps,
02033                                       (gdouble) input->nsteps);
02034             gtk_spin_button_set_value (window->spin_final_steps,
02035                                       (gdouble) input->nfinal_steps);
02036             gtk_spin_button_set_value (window->spin_relaxation,
02037                                       (gdouble) input->relaxation);
02038             switch (input->climbing)
02039             {
02040             case CLIMBING_METHOD_RANDOM:
02041                 gtk_spin_button_set_value (window->spin_estimates,
02042                                           (gdouble) input->nestimates);
02043             }
02044         }
02045         break;
02046     default:
02047         gtk_spin_button_set_value (window->spin_population,
02048                                   (gdouble) input->nsimulations);
02049         gtk_spin_button_set_value (window->spin_generations,
02050                                   (gdouble) input->niterations);
02051         gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
02052         gtk_spin_button_set_value (window->spin_reproduction,
02053                                   input->reproduction_ratio);
02054         gtk_spin_button_set_value (window->spin_adaptation,
02055                                   input->adaptation_ratio);
02056     }
02057     gtk_check_button_set_active (window->button_norm[input->norm], TRUE);
02058     gtk_spin_button_set_value (window->spin_p, input->p);
02059     gtk_spin_button_set_value (window->spin_threshold, input->threshold);
02060     g_signal_handler_block (window->combo_experiment, window->id_experiment);
02061     gtk_combo_box_text_remove_all (window->combo_experiment);
02062     for (i = 0; i < input->nexperiments; ++i)
02063         gtk_combo_box_text_append_text (window->combo_experiment,
02064                                         input->experiment[i].name);
02065     g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
02066     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
02067     g_signal_handler_block (window->combo_variable, window->id_variable);
02068     g_signal_handler_block (window->entry_variable, window->id_variable_label);
02069     gtk_combo_box_text_remove_all (window->combo_variable);
02070     for (i = 0; i < input->nvariables; ++i)
02071         gtk_combo_box_text_append_text (window->combo_variable,
02072                                         input->variable[i].name);
02073     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
02074     g_signal_handler_unblock (window->combo_variable, window->id_variable);
02075     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02076     window_set_variable ();
02077     window_update ();
02078
02079     #if DEBUG_INTERFACE
02080     fprintf (stderr, "window_read: end\n");
02081     #endif
02082     return 1;
02083 }

```

Here is the call graph for this function:



#### 4.11.3.36 window\_remove\_experiment()

```
static void window_remove_experiment ( ) [static]
```

Function to remove an experiment in the main window.

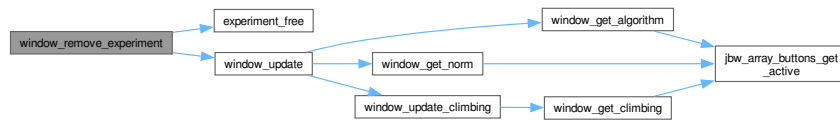
Definition at line 1405 of file [interface.c](#).

```

1406 {
1407     unsigned int i, j;
1408     #if DEBUG_INTERFACE
1409     fprintf (stderr, "window_remove_experiment: start\n");
1410     #endif
1411     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
1412     g_signal_handler_block (window->combo_experiment, window->id_experiment);
1413     gtk_combo_box_text_remove (window->combo_experiment, i);
1414     g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
1415     experiment_free (input->experiment + i, input->type);
1416     --input->nexperiments;
1417     for (j = i; j < input->nexperiments; ++j)
1418         memcpy (input->experiment + j, input->experiment + j + 1,
1419                 sizeof (Experiment));
1420     j = input->nexperiments - 1;
1421     if (i > j)
1422         i = j;
1423     for (j = 0; j < input->experiment->ninputs; ++j)
1424         g_signal_handler_block (window->button_template[j], window->id_input[j]);
1425     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
1426     for (j = 0; j < input->experiment->ninputs; ++j)
1427         g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
1428     window_update ();
1429     #if DEBUG_INTERFACE
1430     fprintf (stderr, "window_remove_experiment: end\n");
1431     #endif
1432 }

```

Here is the call graph for this function:



#### 4.11.3.37 window\_remove\_variable()

```
static void window_remove_variable ( ) [static]
```

Function to remove a variable in the main window.

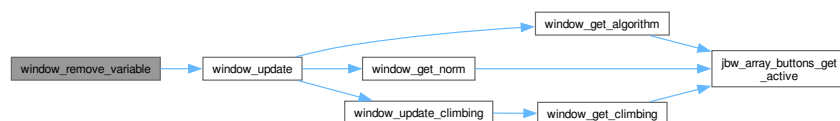
Definition at line 1745 of file [interface.c](#).

```

01746 {
01747     unsigned int i, j;
01748     #if DEBUG_INTERFACE
01749     fprintf (stderr, "window_remove_variable: start\n");
01750     #endif
01751     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01752     g_signal_handler_block (window->combo_variable, window->id_variable);
01753     gtk_combo_box_text_remove (window->combo_variable, i);
01754     g_signal_handler_unblock (window->combo_variable, window->id_variable);
01755     xmlFree (input->variable[i].name);
01756     --input->nvariables;
01757     for (j = i; j < input->nvariables; ++j)
01758         memcpy (input->variable + j, input->variable + j + 1, sizeof (Variable));
01759     j = input->nvariables - 1;
01760     if (i > j)
01761         i = j;
01762     g_signal_handler_block (window->entry_variable, window->id_variable_label);
01763     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01764     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01765     window_update ();
01766     #if DEBUG_INTERFACE
01767     fprintf (stderr, "window_remove_variable: end\n");
01768     #endif
01769 }

```

Here is the call graph for this function:



#### 4.11.3.38 window\_run()

```
static void window_run ( ) [static]
```

Function to run a optimization.

Definition at line 1044 of file [interface.c](#).

```

01045 {
01046     char *msg, *msg2, buffer[64], buffer2[64];

```

Here is the call graph for this function:



Function to save the input file.

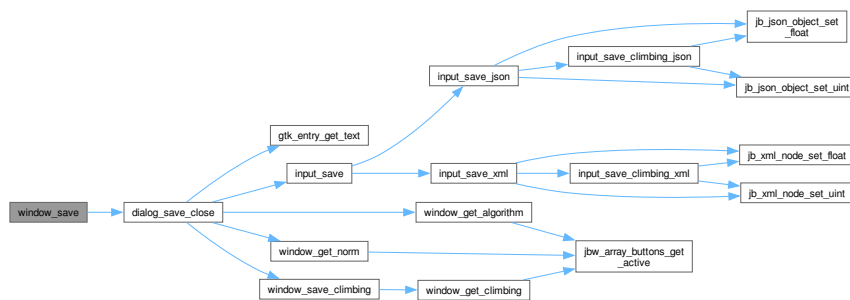
Definition at line 984 of file [interface.c](#).

```

00985 {
00986     GtkFileChooserDialog *dlg;
00987     GtkFileFilter *filter1, *filter2;
00988     char *buffer;
00989
00990 #if DEBUG_INTERFACE
00991     fprintf (stderr, "window_save: start\n");
00992 #endif
00993
00994     // Opening the saving dialog
00995     dlg = (GtkFileChooserDialog *)
00996         gtk_file_chooser_dialog_new (_("Save file"),
00997                                     window->window,
00998                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00999                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
01000                                     _("_OK"), GTK_RESPONSE_OK, NULL);
01001 #if !GTK4
01002     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
01003 #endif
01004     buffer = g_build_filename (input->directory, input->name, NULL);
01005     gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01006     g_free (buffer);
01007
01008     // Adding XML filter
01009     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
01010     gtk_file_filter_set_name (filter1, "XML");
01011     gtk_file_filter_add_pattern (filter1, "*.xml");
01012     gtk_file_filter_add_pattern (filter1, "*.XML");
01013     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
01014
01015     // Adding JSON filter
01016     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
01017     gtk_file_filter_set_name (filter2, "JSON");
01018     gtk_file_filter_add_pattern (filter2, "*.json");
01019     gtk_file_filter_add_pattern (filter2, "*.JSON");
01020     gtk_file_filter_add_pattern (filter2, "*.js");
01021     gtk_file_filter_add_pattern (filter2, "*.JS");
01022     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
01023
01024     if (input->type == INPUT_TYPE_XML)
01025         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
01026     else
01027         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
01028
01029     // Connecting the close function
01030     g_signal_connect (dlg, "response", G_CALLBACK (dialog_save_close), NULL);
01031
01032     // Showing modal dialog
01033     gtk_window_present (GTK_WINDOW (dlg));
01034
01035 #if DEBUG_INTERFACE
01036     fprintf (stderr, "window_save: end\n");
01037 #endif
01038 }

```

Here is the call graph for this function:



#### 4.11.3.40 window\_save\_climbing()

```
static void window_save_climbing ( ) [static]
```



Function to save the hill climbing method data in the input file.

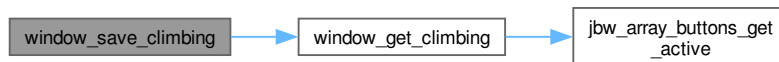
Definition at line 839 of file [interface.c](#).

```

00840 {
00841     #if DEBUG_INTERFACE
00842         fprintf (stderr, "window_save_climbing: start\n");
00843     #endif
00844     if (gtk_check_button_get_active (window->check_climbing))
00845     {
00846         input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
00847         input->nfinal_steps
00848         = gtk_spin_button_get_value_as_int (window->spin_final_steps);
00849         input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
00850         switch (window_get_climbing ())
00851         {
00852             case CLIMBING_METHOD_COORDINATES:
00853                 input->climbing = CLIMBING_METHOD_COORDINATES;
00854                 break;
00855             default:
00856                 input->climbing = CLIMBING_METHOD_RANDOM;
00857                 input->nestimates
00858                 = gtk_spin_button_get_value_as_int (window->spin_estimates);
00859         }
00860     }
00861     else
00862         input->nsteps = 0;
00863     #if DEBUG_INTERFACE
00864         fprintf (stderr, "window_save_climbing: end\n");
00865     #endif
00866 }

```

Here is the call graph for this function:



#### 4.11.3.41 window\_set\_algorithm()

```
static void window_set_algorithm ( ) [static]
```

Function to avoid memory errors changing the algorithm.

Definition at line 1342 of file [interface.c](#).

```

01343 {
01344     int i;
01345     #if DEBUG_INTERFACE
01346         fprintf (stderr, "window_set_algorithm: start\n");
01347     #endif
01348     i = window_get_algorithm ();
01349     switch (i)
01350     {
01351         case ALGORITHM_SWEEP:
01352         case ALGORITHM_ORTHOGONAL:
01353             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01354             if (i < 0)
01355                 i = 0;
01356             gtk_spin_button_set_value (window->spin_sweeps,
01357                                     (gdouble) input->variable[i].nsweeps);
01358             break;
01359         case ALGORITHM_GENETIC:
01360             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01361             if (i < 0)
01362                 i = 0;
01363             gtk_spin_button_set_value (window->spin_bits,
01364                                     (gdouble) input->variable[i].nbits);
01365     }
01366     window_update ();

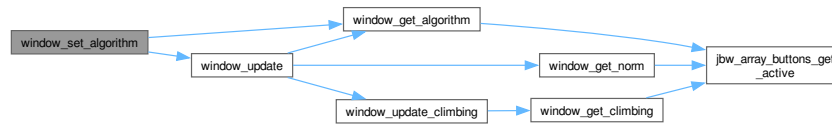
```

```

01367 #if DEBUG_INTERFACE
01368     fprintf (stderr, "window_set_algorithm: end\n");
01369 #endif
01370 }

```

Here is the call graph for this function:



#### 4.11.3.42 window\_set\_experiment()

```
static void window_set_experiment ( ) [static]
```

Function to set the experiment data in the main window.

Definition at line 1376 of file [interface.c](#).

```

01377 {
01378     unsigned int i, j;
01379     char *buffer1;
01380 #if DEBUG_INTERFACE
01381     fprintf (stderr, "window_set_experiment: start\n");
01382 #endif
01383     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01384     gtk_spin_button_set_value (window->spin_weight, input->experiment[i].weight);
01385     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01386     gtk_button_set_label (window->button_experiment, buffer1);
01387     g_free (buffer1);
01388     for (j = 0; j < input->experiment->ninputs; ++j)
01389     {
01390         g_signal_handler_block (window->button_template[j], window->id_input[j]);
01391         gtk_button_set_label (window->button_template[j],
01392                             input->experiment[i].stencil[j]);
01393         g_signal_handler_unblock
01394             (window->button_template[j], window->id_input[j]);
01395     }
01396 #if DEBUG_INTERFACE
01397     fprintf (stderr, "window_set_experiment: end\n");
01398 #endif
01399 }

```

#### 4.11.3.43 window\_set\_variable()

```
static void window_set_variable ( ) [static]
```

Function to set the variable data in the main window.

Definition at line 1670 of file [interface.c](#).

```

01671 {
01672     unsigned int i;
01673 #if DEBUG_INTERFACE
01674     fprintf (stderr, "window_set_variable: start\n");
01675 #endif
01676     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01677     g_signal_handler_block (window->entry_variable, window->id_variable_label);
01678     gtk_entry_set_text (window->entry_variable, input->variable[i].name);
01679     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01680     gtk_spin_button_set_value (window->spin_min, input->variable[i].rangemin);
01681     gtk_spin_button_set_value (window->spin_max, input->variable[i].rangemax);
01682     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01683     {
01684         gtk_spin_button_set_value (window->spin_minabs,

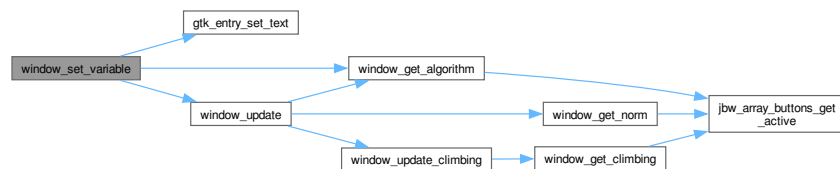
```

```

01685         input->variable[i].rangeminabs);
01686     gtk_check_button_set_active (window->check_minabs, 1);
01687 }
01688 else
01689 {
01690     gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01691     gtk_check_button_set_active (window->check_minabs, 0);
01692 }
01693 if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01694 {
01695     gtk_spin_button_set_value (window->spin_maxabs,
01696         input->variable[i].rangemaxabs);
01697     gtk_check_button_set_active (window->check_maxabs, 1);
01698 }
01699 else
01700 {
01701     gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01702     gtk_check_button_set_active (window->check_maxabs, 0);
01703 }
01704 gtk_spin_button_set_value (window->spin_precision,
01705     input->variable[i].precision);
01706 gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
01707 gtk_spin_button_set_value (window->spin_final_steps,
01708     (gdouble) input->nfinal_steps);
01709 if (input->nsteps)
01710     gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
01711 #if DEBUG_INTERFACE
01712 fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01713     input->variable[i].precision);
01714 #endif
01715 switch (window_get_algorithm ())
01716 {
01717     case ALGORITHM_SWEEP:
01718     case ALGORITHM_ORTHOGONAL:
01719         gtk_spin_button_set_value (window->spin_sweeps,
01720             (gdouble) input->variable[i].nsweeps);
01721 #if DEBUG_INTERFACE
01722 fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01723     input->variable[i].nsweeps);
01724 #endif
01725         break;
01726     case ALGORITHM_GENETIC:
01727         gtk_spin_button_set_value (window->spin_bits,
01728             (gdouble) input->variable[i].nbits);
01729 #if DEBUG_INTERFACE
01730 fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01731     input->variable[i].nbits);
01732 #endif
01733         break;
01734 }
01735 window_update ();
01736 #if DEBUG_INTERFACE
01737 fprintf (stderr, "window_set_variable: end\n");
01738 #endif
01739 }

```

Here is the call graph for this function:



#### 4.11.3.44 window\_step\_variable()

```
static void window_step_variable ( ) [static]
```

Function to update the variable step in the main window.

Definition at line 1927 of file [interface.c](#).

```

01928 {
01929     unsigned int i;
01930     #if DEBUG_INTERFACE
01931     fprintf (stderr, "window_step_variable: start\n");
01932     #endif
01933     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01934     input->variable[i].step = gtk_spin_button_get_value (window->spin_step);
01935     #if DEBUG_INTERFACE
01936     fprintf (stderr, "window_step_variable: end\n");
01937     #endif
01938 }
```

#### 4.11.3.45 window\_template\_experiment()

```

static void window_template_experiment (
    void * data ) [static]
```

Function to update the experiment i-th input template in the main window.

##### Parameters

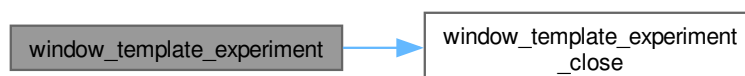
<i>data</i>	Callback data (i-th input template).
-------------	--------------------------------------

Definition at line 1633 of file [interface.c](#).

```

01635 {
01636     GtkFileChooserDialog *dlg;
01637     GMainLoop *loop;
01638     const char *buffer;
01639     unsigned int i;
01640     #if DEBUG_INTERFACE
01641     fprintf (stderr, "window_template_experiment: start\n");
01642     #endif
01643     i = (size_t) data;
01644     buffer = gtk_button_get_label (window->button_template[i]);
01645     dlg = (GtkFileChooserDialog *)
01646         gtk_file_chooser_dialog_new (_("Open template file"),
01647                                     window->window,
01648                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01649                                     _("_Cancel"),
01650                                     GTK_RESPONSE_CANCEL,
01651                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
01652     gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01653     g_signal_connect (dlg, "response",
01654                     G_CALLBACK (window_template_experiment_close), data);
01655     gtk_window_present (GTK_WINDOW (dlg));
01656     loop = g_main_loop_new (NULL, 0);
01657     g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01658                             loop);
01659     g_main_loop_run (loop);
01660     g_main_loop_unref (loop);
01661     #if DEBUG_INTERFACE
01662     fprintf (stderr, "window_template_experiment: end\n");
01663     #endif
01664 }
```

Here is the call graph for this function:



### 4.11.3.46 window\_template\_experiment\_close()

```
static void window_template_experiment_close (
    GtkFileChooserDialog * dlg,
    int response_id,
    void * data ) [static]
```

Function to close the experiment template dialog.

#### Parameters

<i>dlg</i>	GtkFileChooserDialog struct.
<i>response_id</i>	Response identifier.
<i>data</i>	Function data.

Definition at line 1593 of file [interface.c](#).

```
01598 {
01599     GFile *file1, *file2;
01600     char *buffer1, *buffer2;
01601     unsigned int i, j;
01602     #if DEBUG_INTERFACE
01603     fprintf (stderr, "window_template_experiment_close: start\n");
01604     #endif
01605     if (response_id == GTK_RESPONSE_OK)
01606     {
01607         i = (size_t) data;
01608         j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01609         buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01610         file1 = g_file_new_for_path (buffer1);
01611         file2 = g_file_new_for_path (input->directory);
01612         buffer2 = g_file_get_relative_path (file2, file1);
01613         if (input->type == INPUT_TYPE_XML)
01614             input->experiment[j].stencil[i]
01615                 = (char *) xmlStrdup ((xmlChar *) buffer2);
01616         else
01617             input->experiment[j].stencil[i] = g_strdup (buffer2);
01618         g_free (buffer2);
01619         g_object_unref (file2);
01620         g_object_unref (file1);
01621         g_free (buffer1);
01622     }
01623     gtk_window_destroy (GTK_WINDOW (dlg));
01624     #if DEBUG_INTERFACE
01625     fprintf (stderr, "window_template_experiment_close: end\n");
01626     #endif
01627 }
```

### 4.11.3.47 window\_update()

```
static void window_update ( ) [static]
```

Function to update the main window view.

Definition at line 1187 of file [interface.c](#).

```
01188 {
01189     unsigned int i;
01190     #if DEBUG_INTERFACE
01191     fprintf (stderr, "window_update: start\n");
01192     #endif
01193     gtk_widget_set_sensitive
01194         (GTK_WIDGET (window->button_evaluator),
01195          gtk_check_button_get_active (window->check_evaluator));
01196     gtk_widget_set_sensitive
01197         (GTK_WIDGET (window->button_cleaner),
01198          gtk_check_button_get_active (window->check_cleaner));
01199     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01200     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01201     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
```

```

01202 gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01203 gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01204 gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01205 gtk_widget_hide (GTK_WIDGET (window->label_bests));
01206 gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01207 gtk_widget_hide (GTK_WIDGET (window->label_population));
01208 gtk_widget_hide (GTK_WIDGET (window->spin_population));
01209 gtk_widget_hide (GTK_WIDGET (window->label_generations));
01210 gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01211 gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01212 gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01213 gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01214 gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01215 gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01216 gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01217 gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01218 gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01219 gtk_widget_hide (GTK_WIDGET (window->label_bits));
01220 gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01221 gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01222 gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01223 gtk_widget_hide (GTK_WIDGET (window->label_step));
01224 gtk_widget_hide (GTK_WIDGET (window->spin_step));
01225 gtk_widget_hide (GTK_WIDGET (window->label_p));
01226 gtk_widget_hide (GTK_WIDGET (window->spin_p));
01227 i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01228 switch (window_get_algorithm ())
01229 {
01230     case ALGORITHM_MONTE_CARLO:
01231         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01232         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01233         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01234         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01235         if (i > 1)
01236         {
01237             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01238             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01239             gtk_widget_show (GTK_WIDGET (window->label_bests));
01240             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01241         }
01242         window_update_climbing ();
01243         break;
01244     case ALGORITHM_SWEEP:
01245     case ALGORITHM_ORTHOGONAL:
01246         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01247         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01248         if (i > 1)
01249         {
01250             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01251             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01252             gtk_widget_show (GTK_WIDGET (window->label_bests));
01253             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01254         }
01255         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01256         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01257         gtk_widget_show (GTK_WIDGET (window->check_climbing));
01258         window_update_climbing ();
01259         break;
01260     default:
01261         gtk_widget_show (GTK_WIDGET (window->label_population));
01262         gtk_widget_show (GTK_WIDGET (window->spin_population));
01263         gtk_widget_show (GTK_WIDGET (window->label_generations));
01264         gtk_widget_show (GTK_WIDGET (window->spin_generations));
01265         gtk_widget_show (GTK_WIDGET (window->label_mutation));
01266         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01267         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01268         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01269         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01270         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01271         gtk_widget_show (GTK_WIDGET (window->label_bits));
01272         gtk_widget_show (GTK_WIDGET (window->spin_bits));
01273     }
01274 gtk_widget_set_sensitive
01275     (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
01276 gtk_widget_set_sensitive
01277     (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
01278 for (i = 0; i < input->experiment->ninputs; ++i)
01279 {
01280     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01281     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01282     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01283     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01284     g_signal_handler_block
01285         (window->check_template[i], window->id_template[i]);
01286     g_signal_handler_block (window->button_template[i], window->id_input[i]);
01287     gtk_check_button_set_active (window->check_template[i], 1);
01288     g_signal_handler_unblock (window->button_template[i],

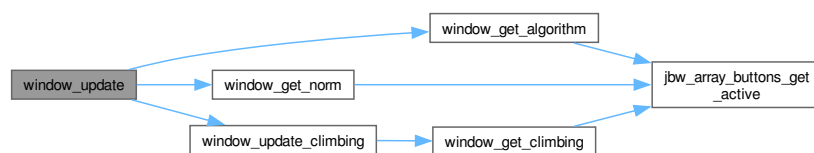
```

```

01289         window->id_input[i]);
01290     g_signal_handler_unblock (window->check_template[i],
01291         window->id_template[i]);
01292 }
01293 if (i > 0)
01294 {
01295     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01296     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01297         gtk_check_button_get_active
01298         (window->check_template[i - 1]));
01299 }
01300 if (i < MAX_NINPUTS)
01301 {
01302     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01303     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01304     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01305     gtk_widget_set_sensitive
01306         (GTK_WIDGET (window->button_template[i]),
01307         gtk_check_button_get_active (window->check_template[i]));
01308     g_signal_handler_block
01309         (window->check_template[i], window->id_template[i]);
01310     g_signal_handler_block (window->button_template[i], window->id_input[i]);
01311     gtk_check_button_set_active (window->check_template[i], 0);
01312     g_signal_handler_unblock (window->button_template[i],
01313         window->id_input[i]);
01314     g_signal_handler_unblock (window->check_template[i],
01315         window->id_template[i]);
01316 }
01317 while (++i < MAX_NINPUTS)
01318 {
01319     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01320     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01321 }
01322 gtk_widget_set_sensitive
01323     (GTK_WIDGET (window->spin_minabs),
01324     gtk_check_button_get_active (window->check_minabs));
01325 gtk_widget_set_sensitive
01326     (GTK_WIDGET (window->spin_maxabs),
01327     gtk_check_button_get_active (window->check_maxabs));
01328 if (window_get_norm () == ERROR_NORM_P)
01329 {
01330     gtk_widget_show (GTK_WIDGET (window->label_p));
01331     gtk_widget_show (GTK_WIDGET (window->spin_p));
01332 }
01333 #if DEBUG_INTERFACE
01334 fprintf (stderr, "window_update: end\n");
01335 #endif
01336 }

```

Here is the call graph for this function:



#### 4.11.3.48 window\_update\_climbing()

```
static void window_update_climbing ( ) [static]
```

Function to update hill climbing method widgets view in the main window.

Definition at line 1156 of file [interface.c](#).

```

01157 {
01158     #if DEBUG_INTERFACE
01159     fprintf (stderr, "window_update_climbing: start\n");
01160     #endif
01161     gtk_widget_show (GTK_WIDGET (window->check_climbing));

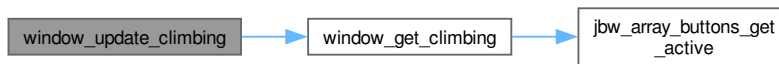
```

```

01162     if (gtk_check_button_get_active (window->check_climbing))
01163     {
01164         gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01165         gtk_widget_show (GTK_WIDGET (window->label_step));
01166         gtk_widget_show (GTK_WIDGET (window->spin_step));
01167     }
01168     switch (window_get_climbing ())
01169     {
01170     case CLIMBING_METHOD_COORDINATES:
01171         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01172         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01173         break;
01174     default:
01175         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01176         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01177     }
01178     #if DEBUG_INTERFACE
01179     fprintf (stderr, "window_update_climbing: end\n");
01180     #endif
01181 }

```

Here is the call graph for this function:



#### 4.11.3.49 window\_update\_variable()

```
static void window_update_variable ( ) [static]
```

Function to update the variable data in the main window.

Definition at line 1944 of file [interface.c](#).

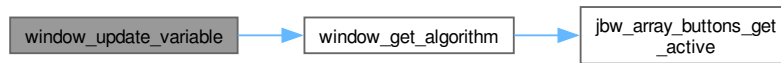
```

01945 {
01946     int i;
01947     #if DEBUG_INTERFACE
01948     fprintf (stderr, "window_update_variable: start\n");
01949     #endif
01950     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01951     if (i < 0)
01952         i = 0;
01953     switch (window_get_algorithm ())
01954     {
01955     case ALGORITHM_SWEEP:
01956     case ALGORITHM_ORTHOGONAL:
01957         input->variable[i].nsweeps
01958         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01959     #if DEBUG_INTERFACE
01960         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01961                 input->variable[i].nsweeps);
01962     #endif
01963         break;
01964     case ALGORITHM_GENETIC:
01965         input->variable[i].nbits
01966         = gtk_spin_button_get_value_as_int (window->spin_bits);
01967     #if DEBUG_INTERFACE
01968         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01969                 input->variable[i].nbits);
01970     #endif
01971     }
01972     #if DEBUG_INTERFACE
01973     fprintf (stderr, "window_update_variable: end\n");
01974     #endif
01975 }

```



Here is the call graph for this function:



#### 4.11.3.50 window\_weight\_experiment()

```
static void window_weight_experiment ( ) [static]
```

Function to update the experiment weight in the main window.

Definition at line 1554 of file [interface.c](#).

```

01555 {
01556     unsigned int i;
01557     #if DEBUG_INTERFACE
01558     fprintf (stderr, "window_weight_experiment: start\n");
01559     #endif
01560     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01561     input->experiment[i].weight = gtk_spin_button_get_value (window->spin_weight);
01562     #if DEBUG_INTERFACE
01563     fprintf (stderr, "window_weight_experiment: end\n");
01564     #endif
01565 }
```

### 4.11.4 Variable Documentation

#### 4.11.4.1 logo

```
const char* logo[] [static]
```

Logo pixmap.

Definition at line 84 of file [interface.c](#).

```

00084 {
00085     "32 32 3 1",
00086     "    c None",
00087     ".    c #0000FF",
00088     "+    c #FF0000",
00089     "
00090     "
00091     "
00092     "    .    .    .    .    "
00093     "    .    .    .    .    "
00094     "    .    .    .    .    "
00095     "    .    .    .    .    "
00096     "    .    .    +++    .    "
00097     "    .    .    +++++    .    "
00098     "    .    .    +++++    .    "
00099     "    .    .    +++++    .    "
00100     "    +++    .    +++    +++    "
00101     "    +++++    .    +++++    "
00102     "    +++++    .    +++++    "
00103     "    +++++    .    +++++    "
00104     "    +++    .    +++    "
00105     "    .    .    .    .    "
00106     "    .    +++    .    .    "
00107     "    .    +++++    .    .    "
00108     "    .    +++++    .    .    "
00109     "    .    +++++    .    .    "
00110     "    .    +++    .    .    "
00111     "    .    .    .    .    "
```

```

00112 " . . . . . ,
00113 " . . . . . ,
00114 " . . . . . ,
00115 " . . . . . ,
00116 " . . . . . ,
00117 " . . . . . ,
00118 " . . . . . ,
00119 " . . . . . ,
00120 " . . . . . ,
00121 };

```

#### 4.11.4.2 options

`Options` options[1] [static]

`Options` struct to define the options dialog.

Definition at line 163 of file [interface.c](#).

#### 4.11.4.3 running

`Running` running[1] [static]

`Running` struct to define the running dialog.

Definition at line 165 of file [interface.c](#).

#### 4.11.4.4 window

`Window` window[1]

`Window` struct to define the main interface window.

Definition at line 81 of file [interface.c](#).

## 4.12 interface.c

[Go to the documentation of this file.](#)

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

```

```

00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #include <gio/gio.h>
00051 #include <gtk/gtk.h>
00052 #include "jb/src/jb_xml.h"
00053 #include "jb/src/jb_json.h"
00054 #include "jb/src/jb_win.h"
00055 #include "genetic/genetic.h"
00056 #include "tools.h"
00057 #include "experiment.h"
00058 #include "variable.h"
00059 #include "input.h"
00060 #include "optimize.h"
00061 #include "interface.h"
00062
00063 #define DEBUG_INTERFACE 1
00064
00065 #ifdef G_OS_WIN32
00066 #define INPUT_FILE "test-ga-win.xml"
00067 #else
00068 #define INPUT_FILE "test-ga.xml"
00069 #endif
00070
00071 Window window[1];
00072
00073 static const char *logo[] = {
00074     "32 32 3 1",
00075     "    c None",
00076     ".    c #0000FF",
00077     "+    c #FF0000",
00078     " ",
00079     " ",
00080     " ",
00081     " ",
00082     ".    .    .    .    .",
00083     ".    .    .    .    .",
00084     ".    .    .    .    .",
00085     ".    .    .    .    .",
00086     ".    .    .    .    .",
00087     ".    .    .    .    .",
00088     ".    .    .    .    .",
00089     ".    .    .    .    .",
00090     ".    .    .    .    .",
00091     ".    .    .    .    .",
00092     ".    .    .    .    .",
00093     ".    .    .    .    .",
00094     ".    .    .    .    .",
00095     ".    .    .    .    .",
00096     ".    .    .    .    .",
00097     ".    .    .    .    .",
00098     ".    .    .    .    .",
00099     ".    .    .    .    .",
00100     ".    .    .    .    .",
00101     ".    .    .    .    .",
00102     ".    .    .    .    .",
00103     ".    .    .    .    .",
00104     ".    .    .    .    .",
00105     ".    .    .    .    .",
00106     ".    .    .    .    .",
00107     ".    .    .    .    .",
00108     ".    .    .    .    .",
00109     ".    .    .    .    .",
00110     ".    .    .    .    .",
00111     ".    .    .    .    .",
00112     ".    .    .    .    .",
00113     ".    .    .    .    .",
00114     ".    .    .    .    .",
00115     ".    .    .    .    .",
00116     ".    .    .    .    .",
00117     ".    .    .    .    .",
00118     ".    .    .    .    .",
00119     ".    .    .    .    .",
00120     ".    .    .    .    .",
00121 };
00122

```

```

00123 /*
00124 const char * logo[] = {
00125 "32 32 3 1",
00126 "      c #FFFFFFFFFFFF",
00127 ".      c #00000000FFFF",
00128 "X      c #FFF00000000",
00129 "                                     ",
00130 "                                     ",
00131 "                                     ",
00132 "      .      .      .      .      ",
00133 "      .      .      .      .      ",
00134 "      .      .      .      .      ",
00135 "      .      .      .      .      ",
00136 "      .      .      XXX      .      ",
00137 "      .      .      XXXXX     .      ",
00138 "      .      .      XXXXX     .      ",
00139 "      .      .      XXXXX     .      ",
00140 "      XXX     .      XXX      XXX     ",
00141 "      XXXXX   .      .      XXXXX   ",
00142 "      XXXXX   .      .      XXXXX   ",
00143 "      XXXXX   .      .      XXXXX   ",
00144 "      XXX     .      .      XXX     ",
00145 "      .      .      .      .      ",
00146 "      .      XXX     .      .      ",
00147 "      .      XXXXX   .      .      ",
00148 "      .      XXXXX   .      .      ",
00149 "      .      XXXXX   .      .      ",
00150 "      .      XXX     .      .      ",
00151 "      .      .      .      .      ",
00152 "      .      .      .      .      ",
00153 "      .      .      .      .      ",
00154 "      .      .      .      .      ",
00155 "      .      .      .      .      ",
00156 "      .      .      .      .      ",
00157 "      .      .      .      .      ",
00158 "      .      .      .      .      ",
00159 "                                     ",
00160 "                                     "};
00161 */
00162
00163 static Options options[1];
00165 static Running running[1];
00167
00171 static void
00172 input_save_climbing_xml (xmlNode * node)
00173 {
00174 #if DEBUG_INTERFACE
00175     fprintf (stderr, "input_save_climbing_xml: start\n");
00176 #endif
00177     if (input->nsteps)
00178     {
00179         jb_xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00180                               input->nsteps);
00181         if (input->relaxation != DEFAULT_RELAXATION)
00182             jb_xml_node_set_float (node, (const xmlChar *) LABEL_RELAXATION,
00183                                   input->relaxation);
00184         switch (input->climbing)
00185         {
00186             case CLIMBING_METHOD_COORDINATES:
00187                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00188                             (const xmlChar *) LABEL_COORDINATES);
00189                 break;
00190             default:
00191                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00192                             (const xmlChar *) LABEL_RANDOM);
00193                 jb_xml_node_set_uint (node, (const xmlChar *) LABEL_NESTIMATES,
00194                                     input->nestimates);
00195         }
00196     }
00197 #if DEBUG_INTERFACE
00198     fprintf (stderr, "input_save_climbing_xml: end\n");
00199 #endif
00200 }
00201
00205 static void
00206 input_save_climbing_json (JsonNode * node)
00207 {
00208     JsonObject *object;
00209 #if DEBUG_INTERFACE
00210     fprintf (stderr, "input_save_climbing_json: start\n");
00211 #endif
00212     object = json_node_get_object (node);
00213     if (input->nsteps)
00214     {
00215         jb_json_object_set_uint (object, LABEL_NSTEPS, input->nsteps);
00216         if (input->relaxation != DEFAULT_RELAXATION)
00217             jb_json_object_set_float (object, LABEL_RELAXATION, input->relaxation);

```

```

00218     switch (input->climbing)
00219     {
00220     case CLIMBING_METHOD_COORDINATES:
00221         json_object_set_string_member (object, LABEL_CLIMBING,
00222                                     LABEL_COORDINATES);
00223         break;
00224     default:
00225         json_object_set_string_member (object, LABEL_CLIMBING, LABEL_RANDOM);
00226         jb_json_object_set_uint (object, LABEL_NESTIMATES, input->nestimates);
00227     }
00228 }
00229 #if DEBUG_INTERFACE
00230 fprintf (stderr, "input_save_climbing_json: end\n");
00231 #endif
00232 }
00233
00234 static inline void
00235 input_save_xml (xmlDoc * doc)
00236 {
00237     unsigned int i, j;
00238     char *buffer;
00239     xmlNode *node, *child;
00240     GFile *file, *file2;
00241
00242     #if DEBUG_INTERFACE
00243     fprintf (stderr, "input_save_xml: start\n");
00244     #endif
00245
00246     // Setting root XML node
00247     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00248     xmlDocSetRootElement (doc, node);
00249
00250     // Adding properties to the root XML node
00251     if (xmlStrcmp
00252         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00253         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00254                   (xmlChar *) input->result);
00255     if (xmlStrcmp
00256         ((const xmlChar *) input->variables, (const xmlChar *) variables_name))
00257         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00258                   (xmlChar *) input->variables);
00259     file = g_file_new_for_path (input->directory);
00260     file2 = g_file_new_for_path (input->simulator);
00261     buffer = g_file_get_relative_path (file, file2);
00262     g_object_unref (file2);
00263     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00264     g_free (buffer);
00265     if (input->evaluator)
00266     {
00267         file2 = g_file_new_for_path (input->evaluator);
00268         buffer = g_file_get_relative_path (file, file2);
00269         g_object_unref (file2);
00270         if (xmlStrlen ((xmlChar *) buffer))
00271             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00272                       (xmlChar *) buffer);
00273         g_free (buffer);
00274     }
00275     if (input->cleaner)
00276     {
00277         file2 = g_file_new_for_path (input->cleaner);
00278         buffer = g_file_get_relative_path (file, file2);
00279         g_object_unref (file2);
00280         if (xmlStrlen ((xmlChar *) buffer))
00281             xmlSetProp (node, (const xmlChar *) LABEL_CLEANER, (xmlChar *) buffer);
00282         g_free (buffer);
00283     }
00284     if (input->seed != DEFAULT_RANDOM_SEED)
00285         jb_xml_node_set_uint (node, (const xmlChar *) LABEL_SEED, input->seed);
00286
00287     // Setting the algorithm
00288     buffer = (char *) g_slice_alloc (64);
00289     switch (input->algorithm)
00290     {
00291     case ALGORITHM_MONTE_CARLO:
00292         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00293                   (const xmlChar *) LABEL_MONTE_CARLO);
00294         snprintf (buffer, 64, "%u", input->nsimulations);
00295         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00296                   (xmlChar *) buffer);
00297         snprintf (buffer, 64, "%u", input->niterations);
00298         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00299                   (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%.3lg", input->tolerance);
00301         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%u", input->nbest);
00303         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304         input_save_climbing_xml (node);

```

```

00308         break;
00309     case ALGORITHM_SWEEP:
00310         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00311                     (const xmlChar *) LABEL_SWEEP);
00312         snprintf (buffer, 64, "%u", input->niterations);
00313         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00314                     (xmlChar *) buffer);
00315         snprintf (buffer, 64, "%.3lg", input->tolerance);
00316         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00317         snprintf (buffer, 64, "%u", input->nbest);
00318         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00319         input_save_climbing_xml (node);
00320         break;
00321     case ALGORITHM_ORTHOGONAL:
00322         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00323                     (const xmlChar *) LABEL_ORTHOGONAL);
00324         snprintf (buffer, 64, "%u", input->niterations);
00325         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00326                     (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%.3lg", input->tolerance);
00328         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00329         snprintf (buffer, 64, "%u", input->nbest);
00330         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00331         input_save_climbing_xml (node);
00332         break;
00333     default:
00334         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00335                     (const xmlChar *) LABEL_GENETIC);
00336         snprintf (buffer, 64, "%u", input->nsimulations);
00337         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00338                     (xmlChar *) buffer);
00339         snprintf (buffer, 64, "%u", input->niterations);
00340         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00341                     (xmlChar *) buffer);
00342         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00343         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00344         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00345         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00346                     (xmlChar *) buffer);
00347         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00348         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00349         break;
00350     }
00351     g_slice_free1 (64, buffer);
00352     if (input->threshold != 0.)
00353         jb_xml_node_set_float (node, (const xmlChar *) LABEL_THRESHOLD,
00354                                input->threshold);
00355
00356     // Setting the experimental data
00357     for (i = 0; i < input->nexperiments; ++i)
00358     {
00359         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00360         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361                     (xmlChar *) input->experiment[i].name);
00362         if (input->experiment[i].weight != 1.)
00363             jb_xml_node_set_float (child, (const xmlChar *) LABEL_WEIGHT,
00364                                    input->experiment[i].weight);
00365         for (j = 0; j < input->experiment->ninputs; ++j)
00366             xmlSetProp (child, (const xmlChar *) stencil[j],
00367                         (xmlChar *) input->experiment[i].stencil[j]);
00368     }
00369
00370     // Setting the variables data
00371     for (i = 0; i < input->nvariables; ++i)
00372     {
00373         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00374         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00375                     (xmlChar *) input->variable[i].name);
00376         jb_xml_node_set_float (child, (const xmlChar *) LABEL_MINIMUM,
00377                                input->variable[i].rangemin);
00378         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00379             jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM,
00380                                    input->variable[i].rangeminabs);
00381         jb_xml_node_set_float (child, (const xmlChar *) LABEL_MAXIMUM,
00382                                input->variable[i].rangemax);
00383         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00384             jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM,
00385                                    input->variable[i].rangemaxabs);
00386         if (input->variable[i].precision != DEFAULT_PRECISION)
00387             jb_xml_node_set_uint (child, (const xmlChar *) LABEL_PRECISION,
00388                                   input->variable[i].precision);
00389         if (input->algorithm == ALGORITHM_SWEEP
00390             || input->algorithm == ALGORITHM_ORTHOGONAL)
00391             jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NSWEEPS,
00392                                   input->variable[i].nsweeps);
00393         else if (input->algorithm == ALGORITHM_GENETIC)
00394             jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,

```

```

00395             input->variable[i].nbits);
00396         if (input->nsteps)
00397             jb_xml_node_set_float (child, (const xmlChar *) LABEL_STEP,
00398             input->variable[i].step);
00399     }
00400
00401     // Saving the error norm
00402     switch (input->norm)
00403     {
00404     case ERROR_NORM_MAXIMUM:
00405         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00406         (const xmlChar *) LABEL_MAXIMUM);
00407         break;
00408     case ERROR_NORM_P:
00409         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00410         (const xmlChar *) LABEL_P);
00411         jb_xml_node_set_float (node, (const xmlChar *) LABEL_P, input->p);
00412         break;
00413     case ERROR_NORM_TAXICAB:
00414         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00415         (const xmlChar *) LABEL_TAXICAB);
00416     }
00417
00418     #if DEBUG_INTERFACE
00419     fprintf (stderr, "input_save: end\n");
00420     #endif
00421 }
00422
00426 static inline void
00427 input_save_json (JsonGenerator * generator)
00428 {
00429     unsigned int i, j;
00430     char *buffer;
00431     XmlNode *node, *child;
00432     JsonObject *object;
00433     JsonArray *array;
00434     GFile *file, *file2;
00435
00436     #if DEBUG_INTERFACE
00437     fprintf (stderr, "input_save_json: start\n");
00438     #endif
00439
00440     // Setting root JSON node
00441     object = json_object_new ();
00442     node = json_node_new (JSON_NODE_OBJECT);
00443     json_node_set_object (node, object);
00444     json_generator_set_root (generator, node);
00445
00446     // Adding properties to the root JSON node
00447     if (strcmp (input->result, result_name))
00448         json_object_set_string_member (object, LABEL_RESULT_FILE, input->result);
00449     if (strcmp (input->variables, variables_name))
00450         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00451         input->variables);
00452     file = g_file_new_for_path (input->directory);
00453     file2 = g_file_new_for_path (input->simulator);
00454     buffer = g_file_get_relative_path (file, file2);
00455     g_object_unref (file2);
00456     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00457     g_free (buffer);
00458     if (input->evaluator)
00459     {
00460         file2 = g_file_new_for_path (input->evaluator);
00461         buffer = g_file_get_relative_path (file, file2);
00462         g_object_unref (file2);
00463         if (strlen (buffer))
00464             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00465         g_free (buffer);
00466     }
00467     if (input->cleaner)
00468     {
00469         file2 = g_file_new_for_path (input->cleaner);
00470         buffer = g_file_get_relative_path (file, file2);
00471         g_object_unref (file2);
00472         if (strlen (buffer))
00473             json_object_set_string_member (object, LABEL_CLEANER, buffer);
00474         g_free (buffer);
00475     }
00476     if (input->seed != DEFAULT_RANDOM_SEED)
00477         jb_json_object_set_uint (object, LABEL_SEED, input->seed);
00478
00479     // Setting the algorithm
00480     buffer = (char *) g_slice_alloc (64);
00481     switch (input->algorithm)
00482     {
00483     case ALGORITHM_MONTE_CARLO:
00484         json_object_set_string_member (object, LABEL_ALGORITHM,

```

```

00485             LABEL_MONTE_CARLO);
00486     snprintf (buffer, 64, "%u", input->nsimulations);
00487     json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00488     snprintf (buffer, 64, "%u", input->niterations);
00489     json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00490     snprintf (buffer, 64, "%.3lg", input->tolerance);
00491     json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00492     snprintf (buffer, 64, "%u", input->nbest);
00493     json_object_set_string_member (object, LABEL_NBEST, buffer);
00494     input_save_climbing_json (node);
00495     break;
00496 case ALGORITHM_SWEEP:
00497     json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_SWEEP);
00498     snprintf (buffer, 64, "%u", input->niterations);
00499     json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00500     snprintf (buffer, 64, "%.3lg", input->tolerance);
00501     json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00502     snprintf (buffer, 64, "%u", input->nbest);
00503     json_object_set_string_member (object, LABEL_NBEST, buffer);
00504     input_save_climbing_json (node);
00505     break;
00506 case ALGORITHM_ORTHOGONAL:
00507     json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_ORTHOGONAL);
00508     snprintf (buffer, 64, "%u", input->niterations);
00509     json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00510     snprintf (buffer, 64, "%.3lg", input->tolerance);
00511     json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00512     snprintf (buffer, 64, "%u", input->nbest);
00513     json_object_set_string_member (object, LABEL_NBEST, buffer);
00514     input_save_climbing_json (node);
00515     break;
00516 default:
00517     json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_GENETIC);
00518     snprintf (buffer, 64, "%u", input->nsimulations);
00519     json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00520     snprintf (buffer, 64, "%u", input->niterations);
00521     json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00522     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00523     json_object_set_string_member (object, LABEL_MUTATION, buffer);
00524     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00525     json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00526     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00527     json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00528     break;
00529 }
00530 g_slice_free1 (64, buffer);
00531 if (input->threshold != 0.)
00532     jb_json_object_set_float (object, LABEL_THRESHOLD, input->threshold);
00533
00534 // Setting the experimental data
00535 array = json_array_new ();
00536 for (i = 0; i < input->nexperiments; ++i)
00537 {
00538     child = json_node_new (JSON_NODE_OBJECT);
00539     object = json_node_get_object (child);
00540     json_object_set_string_member (object, LABEL_NAME,
00541         input->experiment[i].name);
00542     if (input->experiment[i].weight != 1.)
00543         jb_json_object_set_float (object, LABEL_WEIGHT,
00544             input->experiment[i].weight);
00545     for (j = 0; j < input->experiment->ninputs; ++j)
00546         json_object_set_string_member (object, stencil[j],
00547             input->experiment[i].stencil[j]);
00548     json_array_add_element (array, child);
00549 }
00550 json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00551
00552 // Setting the variables data
00553 array = json_array_new ();
00554 for (i = 0; i < input->nvariables; ++i)
00555 {
00556     child = json_node_new (JSON_NODE_OBJECT);
00557     object = json_node_get_object (child);
00558     json_object_set_string_member (object, LABEL_NAME,
00559         input->variable[i].name);
00560     jb_json_object_set_float (object, LABEL_MINIMUM,
00561         input->variable[i].rangemin);
00562     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00563         jb_json_object_set_float (object, LABEL_ABSOLUTE_MINIMUM,
00564             input->variable[i].rangeminabs);
00565     jb_json_object_set_float (object, LABEL_MAXIMUM,
00566         input->variable[i].rangemax);
00567     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00568         jb_json_object_set_float (object, LABEL_ABSOLUTE_MAXIMUM,
00569             input->variable[i].rangemaxabs);
00570     if (input->variable[i].precision != DEFAULT_PRECISION)
00571         jb_json_object_set_uint (object, LABEL_PRECISION,

```



```

00572         input->variable[i].precision);
00573     if (input->algorithm == ALGORITHM_SWEEP
00574         || input->algorithm == ALGORITHM_ORTHOGONAL)
00575         j_b_json_object_set_uint (object, LABEL_NSWEEPS,
00576             input->variable[i].nsweeps);
00577     else if (input->algorithm == ALGORITHM_GENETIC)
00578         j_b_json_object_set_uint (object, LABEL_NBITS, input->variable[i].nbits);
00579     if (input->nsteps)
00580         j_b_json_object_set_float (object, LABEL_STEP, input->variable[i].step);
00581     json_array_add_element (array, child);
00582 }
00583 json_object_set_array_member (object, LABEL_VARIABLES, array);
00584
00585 // Saving the error norm
00586 switch (input->norm)
00587 {
00588     case ERROR_NORM_MAXIMUM:
00589         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00590         break;
00591     case ERROR_NORM_P:
00592         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00593         j_b_json_object_set_float (object, LABEL_P, input->p);
00594         break;
00595     case ERROR_NORM_TAXICAB:
00596         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00597 }
00598
00599 #if DEBUG_INTERFACE
00600 fprintf (stderr, "input_save_json: end\n");
00601 #endif
00602 }
00603
00604 static inline void
00605 input_save (char *filename)
00606 {
00607     xmlDoc *doc;
00608     JsonGenerator *generator;
00609
00610     #if DEBUG_INTERFACE
00611     fprintf (stderr, "input_save: start\n");
00612     #endif
00613
00614     // Getting the input file directory
00615     input->name = g_path_get_basename (filename);
00616     input->directory = g_path_get_dirname (filename);
00617
00618     if (input->type == INPUT_TYPE_XML)
00619     {
00620         // Opening the input file
00621         doc = xmlNewDoc ((const xmlChar *) "1.0");
00622         input_save_xml (doc);
00623
00624         // Saving the XML file
00625         xmlSaveFormatFile (filename, doc, 1);
00626
00627         // Freeing memory
00628         xmlFreeDoc (doc);
00629     }
00630     else
00631     {
00632         // Opening the input file
00633         generator = json_generator_new ();
00634         json_generator_set_pretty (generator, TRUE);
00635         input_save_json (generator);
00636
00637         // Saving the JSON file
00638         json_generator_to_file (generator, filename, NULL);
00639
00640         // Freeing memory
00641         g_object_unref (generator);
00642     }
00643
00644     #if DEBUG_INTERFACE
00645     fprintf (stderr, "input_save: end\n");
00646     #endif
00647 }
00648
00649 static void
00650 dialog_options_close (GtkDialog * dlg,
00651     int response_id)
00652 {
00653     #if DEBUG_INTERFACE
00654     fprintf (stderr, "dialog_options_close: start\n");
00655     #endif
00656     if (response_id == GTK_RESPONSE_OK)
00657     {
00658         input->seed

```

```

00665         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00666         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00667         nthreads_climbing
00668         = gtk_spin_button_get_value_as_int (options->spin_climbing);
00669     }
00670     gtk_window_destroy (GTK_WINDOW (dlg));
00671 #if DEBUG_INTERFACE
00672     fprintf (stderr, "dialog_options_close: end\n");
00673 #endif
00674 }
00675
00676 static void
00677 options_new ()
00678 {
00679     #if DEBUG_INTERFACE
00680     fprintf (stderr, "options_new: start\n");
00681     #endif
00682     options->label_seed = (GtkLabel *)
00683         gtk_label_new (_("Pseudo-random numbers generator seed"));
00684     options->spin_seed = (GtkSpinButton *)
00685         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00686     gtk_widget_set_tooltip_text
00687         (GTK_WIDGET (options->spin_seed),
00688          _("Seed to init the pseudo-random numbers generator"));
00689     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
00690     options->label_threads = (GtkLabel *)
00691         gtk_label_new (_("Threads number for the stochastic algorithm"));
00692     options->spin_threads
00693         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00694     gtk_widget_set_tooltip_text
00695         (GTK_WIDGET (options->spin_threads),
00696          _("Number of threads to perform the calibration/optimization for "
00697           "the stochastic algorithm"));
00698     gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00699     options->label_climbing = (GtkLabel *)
00700         gtk_label_new (_("Threads number for the hill climbing method"));
00701     options->spin_climbing =
00702         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00703     gtk_widget_set_tooltip_text
00704         (GTK_WIDGET (options->spin_climbing),
00705          _("Number of threads to perform the calibration/optimization for the "
00706           "hill climbing method"));
00707     gtk_spin_button_set_value (options->spin_climbing,
00708                               (gdouble) nthreads_climbing);
00709     options->grid = (GtkGrid *) gtk_grid_new ();
00710     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00711     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00712     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00713                     0, 1, 1, 1);
00714     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00715                     1, 1, 1, 1);
00716     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
00717                     1);
00718     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
00719                     1);
00720 #if !GTK4
00721     gtk_widget_show_all (GTK_WIDGET (options->grid));
00722 #else
00723     gtk_widget_show (GTK_WIDGET (options->grid));
00724 #endif
00725     options->dialog = (GtkDialog *)
00726         gtk_dialog_new_with_buttons (_("Options"),
00727                                     window->window,
00728                                     GTK_DIALOG_MODAL,
00729                                     _("_OK"), GTK_RESPONSE_OK,
00730                                     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00731     gtk_box_append (GTK_BOX (gtk_dialog_get_content_area (options->dialog)),
00732                    GTK_WIDGET (options->grid));
00733     g_signal_connect (options->dialog, "response",
00734                      G_CALLBACK (dialog_options_close), NULL);
00735     gtk_window_present (GTK_WINDOW (options->dialog));
00736 #if DEBUG_INTERFACE
00737     fprintf (stderr, "options_new: end\n");
00738 #endif
00739 }
00740
00741 static inline void
00742 running_new ()
00743 {
00744     #if DEBUG_INTERFACE
00745     fprintf (stderr, "running_new: start\n");
00746     #endif
00747     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00748     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00749     running->grid = (GtkGrid *) gtk_grid_new ();
00750     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00751     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);

```

```

00758     running->dialog = (GtkDialog *)
00759         gtk_dialog_new_with_buttons (_, "Calculating"),
00760         window->window, GTK_DIALOG_MODAL, NULL, NULL);
00761     gtk_window_set_child (GTK_WINDOW
00762         (gtk_dialog_get_content_area (running->dialog)),
00763         GTK_WIDGET (running->grid));
00764     gtk_spinner_start (running->spinner);
00765     #if !GTK4
00766     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00767     #else
00768     gtk_widget_show (GTK_WIDGET (running->dialog));
00769     #endif
00770     #if DEBUG_INTERFACE
00771     fprintf (stderr, "running_new: end\n");
00772     #endif
00773 }
00774
00780 static unsigned int
00781 window_get_algorithm ()
00782 {
00783     unsigned int i;
00784     #if DEBUG_INTERFACE
00785     fprintf (stderr, "window_get_algorithm: start\n");
00786     #endif
00787     i = jbw_array_buttons_get_active (window->button_algorithm, NALGORITHMS);
00788     #if DEBUG_INTERFACE
00789     fprintf (stderr, "window_get_algorithm: %u\n", i);
00790     fprintf (stderr, "window_get_algorithm: end\n");
00791     #endif
00792     return i;
00793 }
00794
00800 static unsigned int
00801 window_get_climbing ()
00802 {
00803     unsigned int i;
00804     #if DEBUG_INTERFACE
00805     fprintf (stderr, "window_get_climbing: start\n");
00806     #endif
00807     i = jbw_array_buttons_get_active (window->button_climbing, NCLIMBINGS);
00808     #if DEBUG_INTERFACE
00809     fprintf (stderr, "window_get_climbing: %u\n", i);
00810     fprintf (stderr, "window_get_climbing: end\n");
00811     #endif
00812     return i;
00813 }
00814
00820 static unsigned int
00821 window_get_norm ()
00822 {
00823     unsigned int i;
00824     #if DEBUG_INTERFACE
00825     fprintf (stderr, "window_get_norm: start\n");
00826     #endif
00827     i = jbw_array_buttons_get_active (window->button_norm, NNORMS);
00828     #if DEBUG_INTERFACE
00829     fprintf (stderr, "window_get_norm: %u\n", i);
00830     fprintf (stderr, "window_get_norm: end\n");
00831     #endif
00832     return i;
00833 }
00834
00838 static void
00839 window_save_climbing ()
00840 {
00841     #if DEBUG_INTERFACE
00842     fprintf (stderr, "window_save_climbing: start\n");
00843     #endif
00844     if (gtk_check_button_get_active (window->check_climbing))
00845     {
00846         input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
00847         input->nfinal_steps
00848             = gtk_spin_button_get_value_as_int (window->spin_final_steps);
00849         input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
00850         switch (window_get_climbing ())
00851         {
00852             case CLIMBING_METHOD_COORDINATES:
00853                 input->climbing = CLIMBING_METHOD_COORDINATES;
00854                 break;
00855             default:
00856                 input->climbing = CLIMBING_METHOD_RANDOM;
00857                 input->nestimates
00858                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00859         }
00860     }
00861     else
00862         input->nsteps = 0;

```

```

00863 #if DEBUG_INTERFACE
00864     fprintf (stderr, "window_save_climbing: end\n");
00865 #endif
00866 }
00867
00871 static void
00872 dialog_save_close (GtkFileChooserDialog * dlg,
00873                    int response_id)
00874 {
00875     GtkFileFilter *filter1;
00876     char *buffer;
00877     #if DEBUG_INTERFACE
00878     fprintf (stderr, "dialog_save_close: start\n");
00879     #endif
00881     // If OK response then saving
00882     if (response_id == GTK_RESPONSE_OK)
00883     {
00884         // Setting input file type
00885         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00886         buffer = (char *) gtk_file_filter_get_name (filter1);
00887         if (!strcmp (buffer, "XML"))
00888             input->type = INPUT_TYPE_XML;
00889         else
00890             input->type = INPUT_TYPE_JSON;
00891
00892         // Adding properties to the root XML node
00893         input->simulator
00894             = g_strdup (gtk_button_get_label (window->button_simulator));
00895         if (gtk_check_button_get_active (window->check_evaluator))
00896             input->evaluator
00897                 = g_strdup (gtk_button_get_label (window->button_evaluator));
00898         else
00899             input->evaluator = NULL;
00900         if (gtk_check_button_get_active (window->check_cleaner))
00901             input->cleaner
00902                 = g_strdup (gtk_button_get_label (window->button_cleaner));
00903         else
00904             input->cleaner = NULL;
00905         if (input->type == INPUT_TYPE_XML)
00906         {
00907             input->result
00908                 = (char *) xmlStrdup ((const xmlChar *)
00909                                       gtk_entry_get_text (window->entry_result));
00910             input->variables
00911                 = (char *) xmlStrdup ((const xmlChar *)
00912                                       gtk_entry_get_text (window->entry_variables));
00913         }
00914         else
00915         {
00916             input->result = g_strdup (gtk_entry_get_text (window->entry_result));
00917             input->variables =
00918                 g_strdup (gtk_entry_get_text (window->entry_variables));
00919         }
00920
00921         // Setting the algorithm
00922         switch (window_get_algorithm ())
00923         {
00924             case ALGORITHM_MONTE_CARLO:
00925                 input->algorithm = ALGORITHM_MONTE_CARLO;
00926                 input->nsimulations
00927                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
00928                 input->niterations
00929                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00930                 input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00931                 input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00932                 window_save_climbing ();
00933                 break;
00934             case ALGORITHM_SWEEP:
00935                 input->algorithm = ALGORITHM_SWEEP;
00936                 input->niterations
00937                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00938                 input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00939                 input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00940                 window_save_climbing ();
00941                 break;
00942             case ALGORITHM_ORTHOGONAL:
00943                 input->algorithm = ALGORITHM_ORTHOGONAL;
00944                 input->niterations
00945                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00946                 input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00947                 input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00948                 window_save_climbing ();
00949                 break;
00950             default:
00951                 input->algorithm = ALGORITHM_GENETIC;
00952                 input->nsimulations
00953                     = gtk_spin_button_get_value_as_int (window->spin_population);

```

```

00954         input->niterations
00955         = gtk_spin_button_get_value_as_int (window->spin_generations);
00956         input->mutation_ratio
00957         = gtk_spin_button_get_value (window->spin_mutation);
00958         input->reproduction_ratio
00959         = gtk_spin_button_get_value (window->spin_reproduction);
00960         input->adaptation_ratio
00961         = gtk_spin_button_get_value (window->spin_adaptation);
00962     }
00963     input->norm = window_get_norm ();
00964     input->p = gtk_spin_button_get_value (window->spin_p);
00965     input->threshold = gtk_spin_button_get_value (window->spin_threshold);
00966
00967     // Saving the XML file
00968     buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
00969     input_save (buffer);
00970
00971     // Closing and freeing memory
00972     g_free (buffer);
00973 }
00974 gtk_window_destroy (GTK_WINDOW (dlg));
00975 #if DEBUG_INTERFACE
00976 fprintf (stderr, "dialog_save_close: end\n");
00977 #endif
00978 }
00979
00980 static void
00981 window_save ()
00982 {
00983     GtkFileChooserDialog *dlg;
00984     GtkFileFilter *filter1, *filter2;
00985     char *buffer;
00986
00987     #if DEBUG_INTERFACE
00988         fprintf (stderr, "window_save: start\n");
00989     #endif
00990
00991     // Opening the saving dialog
00992     dlg = (GtkFileChooserDialog *)
00993         gtk_file_chooser_dialog_new (_("Save file"),
00994                                     window->window,
00995                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00996                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00997                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00998     #if !GTK4
00999         gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
01000     #endif
01001     buffer = g_build_filename (input->directory, input->name, NULL);
01002     gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01003     g_free (buffer);
01004
01005     // Adding XML filter
01006     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
01007     gtk_file_filter_set_name (filter1, "XML");
01008     gtk_file_filter_add_pattern (filter1, "*.xml");
01009     gtk_file_filter_add_pattern (filter1, "*.XML");
01010     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
01011
01012     // Adding JSON filter
01013     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
01014     gtk_file_filter_set_name (filter2, "JSON");
01015     gtk_file_filter_add_pattern (filter2, "*.json");
01016     gtk_file_filter_add_pattern (filter2, "*.JSON");
01017     gtk_file_filter_add_pattern (filter2, "*.js");
01018     gtk_file_filter_add_pattern (filter2, "*.JS");
01019     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
01020
01021     if (input->type == INPUT_TYPE_XML)
01022         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
01023     else
01024         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
01025
01026     // Connecting the close function
01027     g_signal_connect (dlg, "response", G_CALLBACK (dialog_save_close), NULL);
01028
01029     // Showing modal dialog
01030     gtk_window_present (GTK_WINDOW (dlg));
01031
01032     #if DEBUG_INTERFACE
01033         fprintf (stderr, "window_save: end\n");
01034     #endif
01035 }
01036
01037 static void
01038 window_run ()
01039 {
01040     char *msg, *msg2, buffer[64], buffer2[64];

```

```

01047     unsigned int i;
01048     #if DEBUG_INTERFACE
01049     fprintf (stderr, "window_run: start\n");
01050     #endif
01051     window_save ();
01052     running_new ();
01053     jbw_process_pending ();
01054     optimize_open ();
01055     #if DEBUG_INTERFACE
01056     fprintf (stderr, "window_run: closing running dialog\n");
01057     #endif
01058     gtk_spinner_stop (running->spinner);
01059     gtk_window_destroy (GTK_WINDOW (running->dialog));
01060     #if DEBUG_INTERFACE
01061     fprintf (stderr, "window_run: displaying results\n");
01062     #endif
01063     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01064     msg2 = g_strdup (buffer);
01065     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01066     {
01067         snprintf (buffer, 64, "%s = %s\n",
01068                 input->variable[i].name, format[input->variable[i].precision]);
01069         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01070         msg = g_strconcat (msg2, buffer2, NULL);
01071         g_free (msg2);
01072     }
01073     snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01074             optimize->calculation_time);
01075     msg = g_strconcat (msg2, buffer, NULL);
01076     g_free (msg2);
01077     jbw_show_message_gtk (_("Best result"), msg, INFO_TYPE);
01078     g_free (msg);
01079     #if DEBUG_INTERFACE
01080     fprintf (stderr, "window_run: freeing memory\n");
01081     #endif
01082     optimize_free ();
01083     #if DEBUG_INTERFACE
01084     fprintf (stderr, "window_run: end\n");
01085     #endif
01086 }
01087
01091 static void
01092 window_help ()
01093 {
01094     char *buffer, *buffer2;
01095     #if DEBUG_INTERFACE
01096     fprintf (stderr, "window_help: start\n");
01097     #endif
01098     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01099                               _("user-manual.pdf"), NULL);
01100     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01101     g_free (buffer2);
01102     #if GTK4
01103     gtk_show_uri (window->window, buffer, GDK_CURRENT_TIME);
01104     #else
01105     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01106     #endif
01107     #if DEBUG_INTERFACE
01108     fprintf (stderr, "window_help: uri=%s\n", buffer);
01109     #endif
01110     g_free (buffer);
01111     #if DEBUG_INTERFACE
01112     fprintf (stderr, "window_help: end\n");
01113     #endif
01114 }
01115
01119 static void
01120 window_about ()
01121 {
01122     static const gchar *authors[] = {
01123         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01124         "Borja Latorre Garcés <borja.latorre@csic.es>",
01125         NULL
01126     };
01127     #if DEBUG_INTERFACE
01128     fprintf (stderr, "window_about: start\n");
01129     #endif
01130     gtk_show_about_dialog
01131     (window->window,
01132      "program_name", "MPCOTool",
01133      "comments",
01134      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01135        "A software to perform calibrations or optimizations of empirical "
01136        "parameters"),
01137      "authors", authors,
01138      "translator-credits",
01139      "Javier Burguete Tolosa <jburguete@eead.csic.es> "

```

```

01140     "(english, french and spanish)\n"
01141     "Uğur Çayoğlu (german)",
01142     "version", "4.10.2",
01143     "copyright", "Copyright 2012-2023 Javier Burguete Tolosa",
01144     "logo", window->logo,
01145     "website", "https://github.com/jburguete/mpcotool",
01146     "license-type", GTK_LICENSE_BSD, NULL);
01147 #if DEBUG_INTERFACE
01148     fprintf (stderr, "window_about: end\n");
01149 #endif
01150 }
01151
01152 static void
01153 window_update_climbing ()
01154 {
01155     #if DEBUG_INTERFACE
01156     fprintf (stderr, "window_update_climbing: start\n");
01157     #endif
01158     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01159     if (gtk_check_button_get_active (window->check_climbing))
01160     {
01161         gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01162         gtk_widget_show (GTK_WIDGET (window->label_step));
01163         gtk_widget_show (GTK_WIDGET (window->spin_step));
01164     }
01165     switch (window_get_climbing ())
01166     {
01167     case CLIMBING_METHOD_COORDINATES:
01168         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01169         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01170         break;
01171     default:
01172         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01173         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01174     }
01175     #if DEBUG_INTERFACE
01176     fprintf (stderr, "window_update_climbing: end\n");
01177     #endif
01178 }
01179
01180 static void
01181 window_update ()
01182 {
01183     unsigned int i;
01184     #if DEBUG_INTERFACE
01185     fprintf (stderr, "window_update: start\n");
01186     #endif
01187     gtk_widget_set_sensitive
01188     (GTK_WIDGET (window->button_evaluator),
01189     gtk_check_button_get_active (window->check_evaluator));
01190     gtk_widget_set_sensitive
01191     (GTK_WIDGET (window->button_cleaner),
01192     gtk_check_button_get_active (window->check_cleaner));
01193     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01194     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01195     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01196     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01197     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01198     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01199     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01200     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01201     gtk_widget_hide (GTK_WIDGET (window->label_population));
01202     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01203     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01204     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01205     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01206     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01207     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01208     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01209     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01210     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01211     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01212     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01213     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01214     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01215     gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01216     gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01217     gtk_widget_hide (GTK_WIDGET (window->label_step));
01218     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01219     gtk_widget_hide (GTK_WIDGET (window->label_p));
01220     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01221     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01222     switch (window_get_algorithm ())
01223     {
01224     case ALGORITHM_MONTE_CARLO:
01225         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01226         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01227     }

```

```

01233     gtk_widget_show (GTK_WIDGET (window->label_iterations));
01234     gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01235     if (i > 1)
01236     {
01237         gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01238         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01239         gtk_widget_show (GTK_WIDGET (window->label_bests));
01240         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01241     }
01242     window_update_climbing ();
01243     break;
01244 case ALGORITHM_SWEEP:
01245 case ALGORITHM_ORTHOGONAL:
01246     gtk_widget_show (GTK_WIDGET (window->label_iterations));
01247     gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01248     if (i > 1)
01249     {
01250         gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01251         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01252         gtk_widget_show (GTK_WIDGET (window->label_bests));
01253         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01254     }
01255     gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01256     gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01257     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01258     window_update_climbing ();
01259     break;
01260 default:
01261     gtk_widget_show (GTK_WIDGET (window->label_population));
01262     gtk_widget_show (GTK_WIDGET (window->spin_population));
01263     gtk_widget_show (GTK_WIDGET (window->label_generations));
01264     gtk_widget_show (GTK_WIDGET (window->spin_generations));
01265     gtk_widget_show (GTK_WIDGET (window->label_mutation));
01266     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01267     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01268     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01269     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01270     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01271     gtk_widget_show (GTK_WIDGET (window->label_bits));
01272     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01273 }
01274 gtk_widget_set_sensitive
01275 (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
01276 gtk_widget_set_sensitive
01277 (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
01278 for (i = 0; i < input->experiment->ninputs; ++i)
01279 {
01280     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01281     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01282     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01283     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01284     g_signal_handler_block
01285     (window->check_template[i], window->id_template[i]);
01286     g_signal_handler_block (window->button_template[i], window->id_input[i]);
01287     gtk_check_button_set_active (window->check_template[i], 1);
01288     g_signal_handler_unblock (window->button_template[i],
01289                             window->id_input[i]);
01290     g_signal_handler_unblock (window->check_template[i],
01291                             window->id_template[i]);
01292 }
01293 if (i > 0)
01294 {
01295     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01296     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01297                             gtk_check_button_get_active
01298                             (window->check_template[i - 1]));
01299 }
01300 if (i < MAX_NINPUTS)
01301 {
01302     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01303     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01304     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01305     gtk_widget_set_sensitive
01306     (GTK_WIDGET (window->button_template[i]),
01307      gtk_check_button_get_active (window->check_template[i]));
01308     g_signal_handler_block
01309     (window->check_template[i], window->id_template[i]);
01310     g_signal_handler_block (window->button_template[i], window->id_input[i]);
01311     gtk_check_button_set_active (window->check_template[i], 0);
01312     g_signal_handler_unblock (window->button_template[i],
01313                             window->id_input[i]);
01314     g_signal_handler_unblock (window->check_template[i],
01315                             window->id_template[i]);
01316 }
01317 while (++i < MAX_NINPUTS)
01318 {
01319     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));

```



```

01320     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01321 }
01322 gtk_widget_set_sensitive
01323 (GTK_WIDGET (window->spin_minabs),
01324  gtk_check_button_get_active (window->check_minabs));
01325 gtk_widget_set_sensitive
01326 (GTK_WIDGET (window->spin_maxabs),
01327  gtk_check_button_get_active (window->check_maxabs));
01328 if (window_get_norm () == ERROR_NORM_P)
01329 {
01330     gtk_widget_show (GTK_WIDGET (window->label_p));
01331     gtk_widget_show (GTK_WIDGET (window->spin_p));
01332 }
01333 #if DEBUG_INTERFACE
01334 fprintf (stderr, "window_update: end\n");
01335 #endif
01336 }
01337
01341 static void
01342 window_set_algorithm ()
01343 {
01344     int i;
01345     #if DEBUG_INTERFACE
01346     fprintf (stderr, "window_set_algorithm: start\n");
01347     #endif
01348     i = window_get_algorithm ();
01349     switch (i)
01350     {
01351     case ALGORITHM_SWEEP:
01352     case ALGORITHM_ORTHOGONAL:
01353         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01354         if (i < 0)
01355             i = 0;
01356         gtk_spin_button_set_value (window->spin_sweeps,
01357                                   (gdouble) input->variable[i].nsweeps);
01358         break;
01359     case ALGORITHM_GENETIC:
01360         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01361         if (i < 0)
01362             i = 0;
01363         gtk_spin_button_set_value (window->spin_bits,
01364                                   (gdouble) input->variable[i].nbits);
01365     }
01366     window_update ();
01367     #if DEBUG_INTERFACE
01368     fprintf (stderr, "window_set_algorithm: end\n");
01369     #endif
01370 }
01371
01375 static void
01376 window_set_experiment ()
01377 {
01378     unsigned int i, j;
01379     char *buffer1;
01380     #if DEBUG_INTERFACE
01381     fprintf (stderr, "window_set_experiment: start\n");
01382     #endif
01383     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01384     gtk_spin_button_set_value (window->spin_weight, input->experiment[i].weight);
01385     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01386     gtk_button_set_label (window->button_experiment, buffer1);
01387     g_free (buffer1);
01388     for (j = 0; j < input->experiment->ninputs; ++j)
01389     {
01390         g_signal_handler_block (window->button_template[j], window->id_input[j]);
01391         gtk_button_set_label (window->button_template[j],
01392                               input->experiment[i].stencil[j]);
01393         g_signal_handler_unblock
01394             (window->button_template[j], window->id_input[j]);
01395     }
01396     #if DEBUG_INTERFACE
01397     fprintf (stderr, "window_set_experiment: end\n");
01398     #endif
01399 }
01400
01404 static void
01405 window_remove_experiment ()
01406 {
01407     unsigned int i, j;
01408     #if DEBUG_INTERFACE
01409     fprintf (stderr, "window_remove_experiment: start\n");
01410     #endif
01411     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01412     g_signal_handler_block (window->combo_experiment, window->id_experiment);
01413     gtk_combo_box_text_remove (window->combo_experiment, i);
01414     g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
01415     experiment_free (input->experiment + i, input->type);

```

```

01416 --input->nexperiments;
01417 for (j = i; j < input->nexperiments; ++j)
01418     memcpy (input->experiment + j, input->experiment + j + 1,
01419             sizeof (Experiment));
01420 j = input->nexperiments - 1;
01421 if (i > j)
01422     i = j;
01423 for (j = 0; j < input->experiment->ninputs; ++j)
01424     g_signal_handler_block (window->button_template[j], window->id_input[j]);
01425 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01426 for (j = 0; j < input->experiment->ninputs; ++j)
01427     g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
01428 window_update ();
01429 #if DEBUG_INTERFACE
01430 fprintf (stderr, "window_remove_experiment: end\n");
01431 #endif
01432 }
01433
01437 static void
01438 window_add_experiment ()
01439 {
01440     unsigned int i, j;
01441     #if DEBUG_INTERFACE
01442     fprintf (stderr, "window_add_experiment: start\n");
01443     #endif
01444     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01445     g_signal_handler_block (window->combo_experiment, window->id_experiment);
01446     gtk_combo_box_text_insert_text
01447         (window->combo_experiment, i, input->experiment[i].name);
01448     g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
01449     input->experiment = (Experiment *) g_realloc
01450         (input->experiment, (input->nexperiments + 1) * sizeof (Experiment));
01451     for (j = input->nexperiments - 1; j > i; --j)
01452         memcpy (input->experiment + j + 1, input->experiment + j,
01453                 sizeof (Experiment));
01454     input->experiment[j + 1].weight = input->experiment[j].weight;
01455     input->experiment[j + 1].ninputs = input->experiment[j].ninputs;
01456     if (input->type == INPUT_TYPE_XML)
01457     {
01458         input->experiment[j + 1].name
01459             = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
01460         for (j = 0; j < input->experiment->ninputs; ++j)
01461             input->experiment[i + 1].stencil[j]
01462                 = (char *) xmlStrdup ((xmlChar *) input->experiment[i].stencil[j]);
01463     }
01464     else
01465     {
01466         input->experiment[j + 1].name = g_strdup (input->experiment[j].name);
01467         for (j = 0; j < input->experiment->ninputs; ++j)
01468             input->experiment[i + 1].stencil[j]
01469                 = g_strdup (input->experiment[i].stencil[j]);
01470     }
01471     ++input->nexperiments;
01472     for (j = 0; j < input->experiment->ninputs; ++j)
01473         g_signal_handler_block (window->button_template[j], window->id_input[j]);
01474     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01475     for (j = 0; j < input->experiment->ninputs; ++j)
01476         g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
01477     window_update ();
01478     #if DEBUG_INTERFACE
01479     fprintf (stderr, "window_add_experiment: end\n");
01480     #endif
01481 }
01482
01486 static void
01487 dialog_name_experiment_close (GtkFileChooserDialog * dlg,
01488                               int response_id,
01489                               void *data)
01490 {
01491     char *buffer;
01492     unsigned int i;
01493     #if DEBUG_INTERFACE
01494     fprintf (stderr, "window_name_experiment_close: start\n");
01495     #endif
01496     if (response_id == GTK_RESPONSE_OK)
01497     {
01498         i = (size_t) data;
01499         buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01500         g_signal_handler_block (window->combo_experiment, window->id_experiment);
01501         gtk_combo_box_text_remove (window->combo_experiment, i);
01502         gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01503         gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01504         g_signal_handler_unblock (window->combo_experiment,
01505                                   window->id_experiment);
01506         g_free (buffer);
01507     }
01508     #if DEBUG_INTERFACE
01509 
```

```

01510     fprintf (stderr, "window_name_experiment_close: end\n");
01511 #endif
01512 }
01513
01514 static void
01515 window_name_experiment ()
01516 {
01517     GtkFileChooserDialog *dlg;
01518     GMainLoop *loop;
01519     const char *buffer;
01520     unsigned int i;
01521 #if DEBUG_INTERFACE
01522     fprintf (stderr, "window_name_experiment: start\n");
01523 #endif
01524     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01525     buffer = gtk_button_get_label (window->button_experiment);
01526     dlg = (GtkFileChooserDialog *)
01527         gtk_file_chooser_dialog_new (_("Open experiment file"),
01528                                     window->window,
01529                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01530                                     _("_Cancel"),
01531                                     GTK_RESPONSE_CANCEL,
01532                                     _("_Open"), GTK_RESPONSE_OK, NULL);
01533     gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01534     g_signal_connect (dlg, "response", G_CALLBACK (dialog_name_experiment_close),
01535                     (void *) (size_t) i);
01536     gtk_window_present (GTK_WINDOW (dlg));
01537     loop = g_main_loop_new (NULL, 0);
01538     g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01539                             loop);
01540     g_main_loop_run (loop);
01541     g_main_loop_unref (loop);
01542 #if DEBUG_INTERFACE
01543     fprintf (stderr, "window_name_experiment: end\n");
01544 #endif
01545 }
01546
01547 static void
01548 window_weight_experiment ()
01549 {
01550     unsigned int i;
01551 #if DEBUG_INTERFACE
01552     fprintf (stderr, "window_weight_experiment: start\n");
01553 #endif
01554     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01555     input->experiment[i].weight = gtk_spin_button_get_value (window->spin_weight);
01556 #if DEBUG_INTERFACE
01557     fprintf (stderr, "window_weight_experiment: end\n");
01558 #endif
01559 }
01560
01561 static void
01562 window_inputs_experiment ()
01563 {
01564     unsigned int j;
01565 #if DEBUG_INTERFACE
01566     fprintf (stderr, "window_inputs_experiment: start\n");
01567 #endif
01568     j = input->experiment->ninputs - 1;
01569     if (j && !gtk_check_button_get_active (window->check_template[j]))
01570         --input->experiment->ninputs;
01571     if (input->experiment->ninputs < MAX_NINPUTS
01572         && gtk_check_button_get_active (window->check_template[j]))
01573         ++input->experiment->ninputs;
01574     window_update ();
01575 #if DEBUG_INTERFACE
01576     fprintf (stderr, "window_inputs_experiment: end\n");
01577 #endif
01578 }
01579
01580 static void
01581 window_template_experiment_close (GtkFileChooserDialog * dlg,
01582                                   int response_id,
01583                                   void *data)
01584 {
01585     GFile *file1, *file2;
01586     char *buffer1, *buffer2;
01587     unsigned int i, j;
01588 #if DEBUG_INTERFACE
01589     fprintf (stderr, "window_template_experiment_close: start\n");
01590 #endif
01591     if (response_id == GTK_RESPONSE_OK)
01592     {
01593         i = (size_t) data;
01594         j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01595         buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01596         file1 = g_file_new_for_path (buffer1);

```

```

01611     file2 = g_file_new_for_path (input->directory);
01612     buffer2 = g_file_get_relative_path (file2, file1);
01613     if (input->type == INPUT_TYPE_XML)
01614         input->experiment[j].stencil[i]
01615         = (char *) xmlStrdup ((xmlChar *) buffer2);
01616     else
01617         input->experiment[j].stencil[i] = g_strdup (buffer2);
01618     g_free (buffer2);
01619     g_object_unref (file2);
01620     g_object_unref (file1);
01621     g_free (buffer1);
01622 }
01623 gtk_window_destroy (GTK_WINDOW (dlg));
01624 #if DEBUG_INTERFACE
01625 fprintf (stderr, "window_template_experiment_close: end\n");
01626 #endif
01627 }
01628
01632 static void
01633 window_template_experiment (void *data)
01634 {
01635     GtkFileChooserDialog *dlg;
01636     GMainLoop *loop;
01637     const char *buffer;
01638     unsigned int i;
01639     #if DEBUG_INTERFACE
01640     fprintf (stderr, "window_template_experiment: start\n");
01641     #endif
01642     i = (size_t) data;
01643     buffer = gtk_button_get_label (window->button_template[i]);
01644     dlg = (GtkFileChooserDialog *)
01645         gtk_file_chooser_dialog_new (_("Open template file"),
01646                                     window->window,
01647                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01648                                     _("_Cancel"),
01649                                     GTK_RESPONSE_CANCEL,
01650                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
01651     gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01652     g_signal_connect (dlg, "response",
01653                      G_CALLBACK (window_template_experiment_close), data);
01654     gtk_window_present (GTK_WINDOW (dlg));
01655     loop = g_main_loop_new (NULL, 0);
01656     g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01657                              loop);
01658     g_main_loop_run (loop);
01659     g_main_loop_unref (loop);
01660     #if DEBUG_INTERFACE
01661     fprintf (stderr, "window_template_experiment: end\n");
01662     #endif
01663 }
01664
01669 static void
01670 window_set_variable ()
01671 {
01672     unsigned int i;
01673     #if DEBUG_INTERFACE
01674     fprintf (stderr, "window_set_variable: start\n");
01675     #endif
01676     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01677     g_signal_handler_block (window->entry_variable, window->id_variable_label);
01678     gtk_entry_set_text (window->entry_variable, input->variable[i].name);
01679     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01680     gtk_spin_button_set_value (window->spin_min, input->variable[i].rangemin);
01681     gtk_spin_button_set_value (window->spin_max, input->variable[i].rangemax);
01682     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01683     {
01684         gtk_spin_button_set_value (window->spin_minabs,
01685                                   input->variable[i].rangeminabs);
01686         gtk_check_button_set_active (window->check_minabs, 1);
01687     }
01688     else
01689     {
01690         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01691         gtk_check_button_set_active (window->check_minabs, 0);
01692     }
01693     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01694     {
01695         gtk_spin_button_set_value (window->spin_maxabs,
01696                                   input->variable[i].rangemaxabs);
01697         gtk_check_button_set_active (window->check_maxabs, 1);
01698     }
01699     else
01700     {
01701         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01702         gtk_check_button_set_active (window->check_maxabs, 0);
01703     }
01704     gtk_spin_button_set_value (window->spin_precision,

```

```

01705         input->variable[i].precision);
01706     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
01707     gtk_spin_button_set_value (window->spin_final_steps,
01708         (gdouble) input->nfinal_steps);
01709     if (input->nsteps)
01710         gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
01711     #if DEBUG_INTERFACE
01712     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01713         input->variable[i].precision);
01714     #endif
01715     switch (window_get_algorithm ())
01716     {
01717     case ALGORITHM_SWEEP:
01718     case ALGORITHM_ORTHOGONAL:
01719         gtk_spin_button_set_value (window->spin_sweeps,
01720             (gdouble) input->variable[i].nsweeps);
01721     #if DEBUG_INTERFACE
01722     fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01723         input->variable[i].nsweeps);
01724     #endif
01725         break;
01726     case ALGORITHM_GENETIC:
01727         gtk_spin_button_set_value (window->spin_bits,
01728             (gdouble) input->variable[i].nbits);
01729     #if DEBUG_INTERFACE
01730     fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01731         input->variable[i].nbits);
01732     #endif
01733         break;
01734     }
01735     window_update ();
01736     #if DEBUG_INTERFACE
01737     fprintf (stderr, "window_set_variable: end\n");
01738     #endif
01739 }
01740
01741 static void
01742 window_remove_variable ()
01743 {
01744     unsigned int i, j;
01745     #if DEBUG_INTERFACE
01746     fprintf (stderr, "window_remove_variable: start\n");
01747     #endif
01748     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01749     g_signal_handler_block (window->combo_variable, window->id_variable);
01750     gtk_combo_box_text_remove (window->combo_variable, i);
01751     g_signal_handler_unblock (window->combo_variable, window->id_variable);
01752     xmlFree (input->variable[i].name);
01753     --input->nvariables;
01754     for (j = i; j < input->nvariables; ++j)
01755         memcpy (input->variable + j, input->variable + j + 1, sizeof (Variable));
01756     j = input->nvariables - 1;
01757     if (i > j)
01758         i = j;
01759     g_signal_handler_block (window->entry_variable, window->id_variable_label);
01760     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01761     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01762     window_update ();
01763     #if DEBUG_INTERFACE
01764     fprintf (stderr, "window_remove_variable: end\n");
01765     #endif
01766 }
01767
01768 static void
01769 window_add_variable ()
01770 {
01771     unsigned int i, j;
01772     #if DEBUG_INTERFACE
01773     fprintf (stderr, "window_add_variable: start\n");
01774     #endif
01775     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01776     g_signal_handler_block (window->combo_variable, window->id_variable);
01777     gtk_combo_box_text_insert_text (window->combo_variable, i,
01778         input->variable[i].name);
01779     g_signal_handler_unblock (window->combo_variable, window->id_variable);
01780     input->variable = (Variable *) g_realloc
01781         (input->variable, (input->nvariables + 1) * sizeof (Variable));
01782     for (j = input->nvariables - 1; j > i; --j)
01783         memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01784     memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01785     if (input->type == INPUT_TYPE_XML)
01786         input->variable[j + 1].name
01787             = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01788     else
01789         input->variable[j + 1].name = g_strdup (input->variable[j].name);
01790     ++input->nvariables;
01791     g_signal_handler_block (window->entry_variable, window->id_variable_label);

```

```

01798     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01799     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01800     window_update ();
01801     #if DEBUG_INTERFACE
01802     fprintf (stderr, "window_add_variable: end\n");
01803     #endif
01804 }
01805
01806 static void
01807 window_label_variable ()
01808 {
01809     unsigned int i;
01810     const char *buffer;
01811     #if DEBUG_INTERFACE
01812     fprintf (stderr, "window_label_variable: start\n");
01813     #endif
01814     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01815     buffer = gtk_entry_get_text (window->entry_variable);
01816     g_signal_handler_block (window->combo_variable, window->id_variable);
01817     gtk_combo_box_text_remove (window->combo_variable, i);
01818     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01819     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01820     g_signal_handler_unblock (window->combo_variable, window->id_variable);
01821     #if DEBUG_INTERFACE
01822     fprintf (stderr, "window_label_variable: end\n");
01823     #endif
01824 }
01825
01826 static void
01827 window_precision_variable ()
01828 {
01829     unsigned int i;
01830     #if DEBUG_INTERFACE
01831     fprintf (stderr, "window_precision_variable: start\n");
01832     #endif
01833     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01834     input->variable[i].precision
01835     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01836     gtk_spin_button_set_digits (window->spin_min, input->variable[i].precision);
01837     gtk_spin_button_set_digits (window->spin_max, input->variable[i].precision);
01838     gtk_spin_button_set_digits (window->spin_minabs,
01839                                input->variable[i].precision);
01840     gtk_spin_button_set_digits (window->spin_maxabs,
01841                                input->variable[i].precision);
01842     #if DEBUG_INTERFACE
01843     fprintf (stderr, "window_precision_variable: end\n");
01844     #endif
01845 }
01846
01847 static void
01848 window_rangemin_variable ()
01849 {
01850     unsigned int i;
01851     #if DEBUG_INTERFACE
01852     fprintf (stderr, "window_rangemin_variable: start\n");
01853     #endif
01854     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01855     input->variable[i].rangemin = gtk_spin_button_get_value (window->spin_min);
01856     #if DEBUG_INTERFACE
01857     fprintf (stderr, "window_rangemin_variable: end\n");
01858     #endif
01859 }
01860
01861 static void
01862 window_rangemax_variable ()
01863 {
01864     unsigned int i;
01865     #if DEBUG_INTERFACE
01866     fprintf (stderr, "window_rangemax_variable: start\n");
01867     #endif
01868     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01869     input->variable[i].rangemax = gtk_spin_button_get_value (window->spin_max);
01870     #if DEBUG_INTERFACE
01871     fprintf (stderr, "window_rangemax_variable: end\n");
01872     #endif
01873 }
01874
01875 static void
01876 window_rangeminabs_variable ()
01877 {
01878     unsigned int i;
01879     #if DEBUG_INTERFACE
01880     fprintf (stderr, "window_rangeminabs_variable: start\n");
01881     #endif
01882     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01883     input->variable[i].rangeminabs
01884     = gtk_spin_button_get_value (window->spin_minabs);
01885 }

```

```

01900 #if DEBUG_INTERFACE
01901     fprintf (stderr, "window_rangeminabs_variable: end\n");
01902 #endif
01903 }
01904
01908 static void
01909 window_rangemaxabs_variable ()
01910 {
01911     unsigned int i;
01912 #if DEBUG_INTERFACE
01913     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01914 #endif
01915     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01916     input->variable[i].rangemaxabs
01917         = gtk_spin_button_get_value (window->spin_maxabs);
01918 #if DEBUG_INTERFACE
01919     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01920 #endif
01921 }
01922
01926 static void
01927 window_step_variable ()
01928 {
01929     unsigned int i;
01930 #if DEBUG_INTERFACE
01931     fprintf (stderr, "window_step_variable: start\n");
01932 #endif
01933     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01934     input->variable[i].step = gtk_spin_button_get_value (window->spin_step);
01935 #if DEBUG_INTERFACE
01936     fprintf (stderr, "window_step_variable: end\n");
01937 #endif
01938 }
01939
01943 static void
01944 window_update_variable ()
01945 {
01946     int i;
01947 #if DEBUG_INTERFACE
01948     fprintf (stderr, "window_update_variable: start\n");
01949 #endif
01950     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01951     if (i < 0)
01952         i = 0;
01953     switch (window_get_algorithm ())
01954     {
01955         case ALGORITHM_SWEEP:
01956         case ALGORITHM_ORTHOGONAL:
01957             input->variable[i].nsweeps
01958                 = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01959 #if DEBUG_INTERFACE
01960             fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01961                     input->variable[i].nsweeps);
01962 #endif
01963             break;
01964         case ALGORITHM_GENETIC:
01965             input->variable[i].nbits
01966                 = gtk_spin_button_get_value_as_int (window->spin_bits);
01967 #if DEBUG_INTERFACE
01968             fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01969                     input->variable[i].nbits);
01970 #endif
01971     }
01972 #if DEBUG_INTERFACE
01973     fprintf (stderr, "window_update_variable: end\n");
01974 #endif
01975 }
01976
01982 static int
01983 window_read (char *filename)
01984 {
01985     unsigned int i;
01986 #if DEBUG_INTERFACE
01987     fprintf (stderr, "window_read: start\n");
01988     fprintf (stderr, "window_read: file name=%s\n", filename);
01989 #endif
01990
01991     // Reading new input file
01992     input_free ();
01993     input->result = input->variables = NULL;
01994     if (!input_open (filename))
01995     {
01996 #if DEBUG_INTERFACE
01997         fprintf (stderr, "window_read: end\n");
01998 #endif
01999         return 0;
02000     }

```

```

02001
02002 // Setting GTK+ widgets data
02003 gtk_entry_set_text (window->entry_result, input->result);
02004 gtk_entry_set_text (window->entry_variables, input->variables);
02005 gtk_button_set_label (window->button_simulator, input->simulator);
02006 gtk_check_button_set_active (window->check_evaluator,
02007                             (size_t) input->evaluator);
02008 if (input->evaluator)
02009     gtk_button_set_label (window->button_evaluator, input->evaluator);
02010 gtk_check_button_set_active (window->check_cleaner, (size_t) input->cleaner);
02011 if (input->cleaner)
02012     gtk_button_set_label (window->button_cleaner, input->cleaner);
02013 gtk_check_button_set_active (window->button_algorithm[input->algorithm],
02014                             TRUE);
02015 switch (input->algorithm)
02016 {
02017     case ALGORITHM_MONTE_CARLO:
02018         gtk_spin_button_set_value (window->spin_simulations,
02019                                   (gdouble) input->nsimulations);
02020         // fallthrough
02021     case ALGORITHM_SWEEP:
02022     case ALGORITHM_ORTHOGONAL:
02023         gtk_spin_button_set_value (window->spin_iterations,
02024                                   (gdouble) input->niterations);
02025         gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
02026         gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
02027         gtk_check_button_set_active (window->check_climbing, input->nsteps);
02028         if (input->nsteps)
02029         {
02030             gtk_check_button_set_active
02031                 (window->button_climbing[input->climbing], TRUE);
02032             gtk_spin_button_set_value (window->spin_steps,
02033                                       (gdouble) input->nsteps);
02034             gtk_spin_button_set_value (window->spin_final_steps,
02035                                       (gdouble) input->nfinal_steps);
02036             gtk_spin_button_set_value (window->spin_relaxation,
02037                                       (gdouble) input->relaxation);
02038             switch (input->climbing)
02039             {
02040                 case CLIMBING_METHOD_RANDOM:
02041                     gtk_spin_button_set_value (window->spin_estimates,
02042                                                 (gdouble) input->nestimates);
02043             }
02044         }
02045         break;
02046     default:
02047         gtk_spin_button_set_value (window->spin_population,
02048                                   (gdouble) input->nsimulations);
02049         gtk_spin_button_set_value (window->spin_generations,
02050                                   (gdouble) input->niterations);
02051         gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
02052         gtk_spin_button_set_value (window->spin_reproduction,
02053                                   input->reproduction_ratio);
02054         gtk_spin_button_set_value (window->spin_adaptation,
02055                                   input->adaptation_ratio);
02056     }
02057     gtk_check_button_set_active (window->button_norm[input->norm], TRUE);
02058     gtk_spin_button_set_value (window->spin_p, input->p);
02059     gtk_spin_button_set_value (window->spin_threshold, input->threshold);
02060     g_signal_handler_block (window->combo_experiment, window->id_experiment);
02061     gtk_combo_box_text_remove_all (window->combo_experiment);
02062     for (i = 0; i < input->nexperiments; ++i)
02063         gtk_combo_box_text_append_text (window->combo_experiment,
02064                                         input->experiment[i].name);
02065     g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
02066     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
02067     g_signal_handler_block (window->combo_variable, window->id_variable);
02068     g_signal_handler_block (window->entry_variable, window->id_variable_label);
02069     gtk_combo_box_text_remove_all (window->combo_variable);
02070     for (i = 0; i < input->nvariables; ++i)
02071         gtk_combo_box_text_append_text (window->combo_variable,
02072                                         input->variable[i].name);
02073     g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
02074     g_signal_handler_unblock (window->combo_variable, window->id_variable);
02075     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02076     window_set_variable ();
02077     window_update ();
02078
02079 #if DEBUG_INTERFACE
02080     fprintf (stderr, "window_read: end\n");
02081 #endif
02082     return 1;
02083 }
02084
02088 static void
02089 dialog_open_close (GtkFileChooserDialog * dlg,
02090                   int response_id)

```



```

02092 {
02093     char *buffer, *directory, *name;
02094     GFile *file;
02095
02096     #if DEBUG_INTERFACE
02097         fprintf (stderr, "dialog_open_close: start\n");
02098     #endif
02099
02100     // Saving a backup of the current input file
02101     directory = g_strdup (input->directory);
02102     name = g_strdup (input->name);
02103
02104     // If OK saving
02105     if (response_id == GTK_RESPONSE_OK)
02106     {
02107         // Trying to open the input file
02108         file = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (dlg));
02109         buffer = g_file_get_path (file);
02110     #if DEBUG_INTERFACE
02111         fprintf (stderr, "dialog_open_close: file name=%s\n", buffer);
02112     #endif
02113     #if !window_read (buffer)
02114     {
02115         #if DEBUG_INTERFACE
02116             fprintf (stderr, "dialog_open_close: error reading input file\n");
02117         #endif
02118         g_free (buffer);
02119
02120         // Reading backup file on error
02121         buffer = g_build_filename (directory, name, NULL);
02122         input->result = input->variables = NULL;
02123         if (!input_open (buffer))
02124         {
02125             // Closing on backup file reading error
02126             #if DEBUG_INTERFACE
02127                 fprintf (stderr,
02128                     "dialog_open_close: error reading backup file\n");
02129             #endif
02130         }
02131     }
02132     g_free (buffer);
02133     g_object_unref (file);
02134
02135     // Freeing and closing
02136     g_free (name);
02137     g_free (directory);
02138     gtk_window_destroy (GTK_WINDOW (dlg));
02139
02140     #if DEBUG_INTERFACE
02141         fprintf (stderr, "dialog_open_close: end\n");
02142     #endif
02143 }
02144
02145 static void
02146 window_open ()
02147 {
02148     GtkFileChooserDialog *dlg;
02149     GtkFileFilter *filter;
02150
02151     #if DEBUG_INTERFACE
02152         fprintf (stderr, "window_open: start\n");
02153     #endif
02154
02155     // Opening dialog
02156     dlg = (GtkFileChooserDialog *)
02157         gtk_file_chooser_dialog_new (_("Open input file"),
02158             window->window,
02159             GTK_FILE_CHOOSER_ACTION_OPEN,
02160             _("_Cancel"), GTK_RESPONSE_CANCEL,
02161             _("_OK"), GTK_RESPONSE_OK, NULL);
02162
02163     // Adding XML filter
02164     filter = (GtkFileFilter *) gtk_file_filter_new ();
02165     gtk_file_filter_set_name (filter, "XML");
02166     gtk_file_filter_add_pattern (filter, "*.xml");
02167     gtk_file_filter_add_pattern (filter, "*.XML");
02168     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02169
02170     // Adding JSON filter
02171     filter = (GtkFileFilter *) gtk_file_filter_new ();
02172     gtk_file_filter_set_name (filter, "JSON");
02173     gtk_file_filter_add_pattern (filter, "*.json");
02174     gtk_file_filter_add_pattern (filter, "*.JSON");
02175 }

```

```

02182 gtk_file_filter_add_pattern (filter, "*.js");
02183 gtk_file_filter_add_pattern (filter, "*.JS");
02184 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02185
02186 // Connecting the close function
02187 g_signal_connect (dlg, "response", G_CALLBACK (dialog_open_close), NULL);
02188
02189 // Showing modal dialog
02190 gtk_window_present (GTK_WINDOW (dlg));
02191
02192 #if DEBUG_INTERFACE
02193 fprintf (stderr, "window_open: end\n");
02194 #endif
02195 }
02196
02200 static void
02201 dialog_simulator_close (GtkFileChooserDialog * dlg,
02202 int response_id)
02203 {
02204 {
02205 GFile *file1, *file2;
02206 char *buffer1, *buffer2;
02207 #if DEBUG_INTERFACE
02208 fprintf (stderr, "dialog_simulator_close: start\n");
02209 #endif
02210 if (response_id == GTK_RESPONSE_OK)
02211 {
02212 buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02213 file1 = g_file_new_for_path (buffer1);
02214 file2 = g_file_new_for_path (input->directory);
02215 buffer2 = g_file_get_relative_path (file2, file1);
02216 input->simulator = g_strdup (buffer2);
02217 g_free (buffer2);
02218 g_object_unref (file2);
02219 g_object_unref (file1);
02220 g_free (buffer1);
02221 }
02222 gtk_window_destroy (GTK_WINDOW (dlg));
02223 #if DEBUG_INTERFACE
02224 fprintf (stderr, "dialog_simulator_close: end\n");
02225 #endif
02226 }
02227
02231 static void
02232 dialog_simulator ()
02233 {
02234 GtkFileChooserDialog *dlg;
02235 #if DEBUG_INTERFACE
02236 fprintf (stderr, "dialog_simulator: start\n");
02237 #endif
02238 dlg = (GtkFileChooserDialog *)
02239 gtk_file_chooser_dialog_new (_("Open simulator file"),
02240 window->window,
02241 GTK_FILE_CHOOSER_ACTION_OPEN,
02242 _("_Cancel"), GTK_RESPONSE_CANCEL,
02243 _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02244 g_signal_connect (dlg, "response", G_CALLBACK (dialog_simulator_close), NULL);
02245 gtk_window_present (GTK_WINDOW (dlg));
02246 #if DEBUG_INTERFACE
02247 fprintf (stderr, "dialog_simulator: end\n");
02248 #endif
02249 }
02250
02254 static void
02255 dialog_evaluator_close (GtkFileChooserDialog * dlg,
02256 int response_id)
02257 {
02258 {
02259 GFile *file1, *file2;
02260 char *buffer1, *buffer2;
02261 #if DEBUG_INTERFACE
02262 fprintf (stderr, "dialog_evaluator_close: start\n");
02263 #endif
02264 if (response_id == GTK_RESPONSE_OK)
02265 {
02266 buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02267 file1 = g_file_new_for_path (buffer1);
02268 file2 = g_file_new_for_path (input->directory);
02269 buffer2 = g_file_get_relative_path (file2, file1);
02270 input->evaluator = g_strdup (buffer2);
02271 g_free (buffer2);
02272 g_object_unref (file2);
02273 g_object_unref (file1);
02274 g_free (buffer1);
02275 }
02276 gtk_window_destroy (GTK_WINDOW (dlg));
02277 #if DEBUG_INTERFACE
02278 fprintf (stderr, "dialog_evaluator_close: end\n");
02279 #endif

```

```

02280 }
02281
02285 static void
02286 dialog_evaluator ()
02287 {
02288     GtkFileChooserDialog *dlg;
02289     #if DEBUG_INTERFACE
02290     fprintf (stderr, "dialog_evaluator: start\n");
02291     #endif
02292     dlg = (GtkFileChooserDialog *)
02293         gtk_file_chooser_dialog_new (_("Open evaluator file"),
02294                                     window->window,
02295                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02296                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02297                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02298     g_signal_connect (dlg, "response", G_CALLBACK (dialog_evaluator_close), NULL);
02299     gtk_window_present (GTK_WINDOW (dlg));
02300     #if DEBUG_INTERFACE
02301     fprintf (stderr, "dialog_evaluator: end\n");
02302     #endif
02303 }
02304
02308 static void
02309 dialog_cleaner_close (GtkFileChooserDialog * dlg,
02310                      int response_id)
02311 {
02312     GFile *file1, *file2;
02313     char *buffer1, *buffer2;
02314     #if DEBUG_INTERFACE
02315     fprintf (stderr, "dialog_cleaner_close: start\n");
02316     #endif
02317     if (response_id == GTK_RESPONSE_OK)
02318     {
02319         buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02320         file1 = g_file_new_for_path (buffer1);
02321         file2 = g_file_new_for_path (input->directory);
02322         buffer2 = g_file_get_relative_path (file2, file1);
02323         input->cleaner = g_strdup (buffer2);
02324         g_free (buffer2);
02325         g_object_unref (file2);
02326         g_object_unref (file1);
02327         g_free (buffer1);
02328     }
02329     gtk_window_destroy (GTK_WINDOW (dlg));
02330     #if DEBUG_INTERFACE
02331     fprintf (stderr, "dialog_cleaner_close: end\n");
02332     #endif
02333 }
02334
02339 static void
02340 dialog_cleaner ()
02341 {
02342     GtkFileChooserDialog *dlg;
02343     #if DEBUG_INTERFACE
02344     fprintf (stderr, "dialog_cleaner: start\n");
02345     #endif
02346     dlg = (GtkFileChooserDialog *)
02347         gtk_file_chooser_dialog_new (_("Open cleaner file"),
02348                                     window->window,
02349                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02350                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02351                                     _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02352     g_signal_connect (dlg, "response", G_CALLBACK (dialog_cleaner_close), NULL);
02353     gtk_window_present (GTK_WINDOW (dlg));
02354     #if DEBUG_INTERFACE
02355     fprintf (stderr, "dialog_cleaner: end\n");
02356     #endif
02357 }
02358
02362 void
02363 window_new (GtkApplication * application)
02364 {
02365     unsigned int i;
02366     char *buffer, *buffer2, buffer3[64];
02367     const char *label_algorithm[NALGORITHMS] = {
02368         "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02369     };
02370     const char *tip_algorithm[NALGORITHMS] = {
02371         _("_Monte-Carlo brute force algorithm"),
02372         _("_Sweep brute force algorithm"),
02373         _("_Genetic algorithm"),
02374         _("_Orthogonal sampling brute force algorithm"),
02375     };
02376     const char *label_climbing[NCLIMBINGS] = {
02377         _("_Coordinates climbing"), _("_Random climbing")
02378     };
02379     const char *tip_climbing[NCLIMBINGS] = {

```

```

02380     _("Coordinates climbing estimate method"),
02381     _("Random climbing estimate method")
02382 };
02383 const char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02384 const char *tip_norm[NNORMS] = {
02385     _("Euclidean error norm (L2)"),
02386     _("Maximum error norm (L)"),
02387     _("P error norm (Lp)"),
02388     _("Taxicab error norm (L1)")
02389 };
02390 #if !GTK4
02391     const char *close = "delete-event";
02392 #else
02393     const char *close = "close-request";
02394 #endif
02395
02396 #if DEBUG_INTERFACE
02397     fprintf(stderr, "window_new: start\n");
02398 #endif
02399
02400     // Creating the window
02401     window->window = window_parent = main_window
02402     = (GtkWindow *) gtk_application_window_new (application);
02403
02404     // Finish when closing the window
02405     g_signal_connect_swapped (window->window, close,
02406         G_CALLBACK (g_application_quit),
02407         G_APPLICATION (application));
02408
02409     // Setting the window title
02410     gtk_window_set_title (window->window, "MPCOTool");
02411
02412     // Creating the open button
02413     window->button_open = (GtkButton *)
02414 #if !GTK4
02415     gtk_button_new_from_icon_name ("document-open", GTK_ICON_SIZE_BUTTON);
02416 #else
02417     gtk_button_new_from_icon_name ("document-open");
02418 #endif
02419     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_open),
02420         _("Open a case"));
02421     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02422
02423     // Creating the save button
02424     window->button_save = (GtkButton *)
02425 #if !GTK4
02426     gtk_button_new_from_icon_name ("document-save", GTK_ICON_SIZE_BUTTON);
02427 #else
02428     gtk_button_new_from_icon_name ("document-save");
02429 #endif
02430     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_save),
02431         _("Save the case"));
02432     g_signal_connect (window->button_save, "clicked", (GCallback) window_save,
02433         NULL);
02434
02435     // Creating the run button
02436     window->button_run = (GtkButton *)
02437 #if !GTK4
02438     gtk_button_new_from_icon_name ("system-run", GTK_ICON_SIZE_BUTTON);
02439 #else
02440     gtk_button_new_from_icon_name ("system-run");
02441 #endif
02442     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_run),
02443         _("Run the optimization"));
02444     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02445
02446     // Creating the options button
02447     window->button_options = (GtkButton *)
02448 #if !GTK4
02449     gtk_button_new_from_icon_name ("preferences-system", GTK_ICON_SIZE_BUTTON);
02450 #else
02451     gtk_button_new_from_icon_name ("preferences-system");
02452 #endif
02453     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_options),
02454         _("Edit the case"));
02455     g_signal_connect (window->button_options, "clicked", options_new, NULL);
02456
02457     // Creating the help button
02458     window->button_help = (GtkButton *)
02459 #if !GTK4
02460     gtk_button_new_from_icon_name ("help-browser", GTK_ICON_SIZE_BUTTON);
02461 #else
02462     gtk_button_new_from_icon_name ("help-browser");
02463 #endif
02464     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_help), _("Help"));
02465     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02466

```

```

02467 // Creating the about button
02468 window->button_about = (GtkButton *)
02469 #if !GTK4
02470     gtk_button_new_from_icon_name ("help-about", GTK_ICON_SIZE_BUTTON);
02471 #else
02472     gtk_button_new_from_icon_name ("help-about");
02473 #endif
02474 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_about), _("About"));
02475 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02476
02477 // Creating the exit button
02478 window->button_exit = (GtkButton *)
02479 #if !GTK4
02480     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
02481 #else
02482     gtk_button_new_from_icon_name ("application-exit");
02483 #endif
02484 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_exit), _("Exit"));
02485 g_signal_connect_swapped (window->button_exit, "clicked",
02486                           G_CALLBACK (g_application_quit),
02487                           G_APPLICATION (application));
02488
02489 // Creating the buttons bar
02490 window->box_buttons = (GtkBox *) gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
02491 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_open));
02492 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_save));
02493 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_run));
02494 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_options));
02495 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_help));
02496 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_about));
02497 gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_exit));
02498
02499 // Creating the simulator program label and entry
02500 window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02501 window->button_simulator = (GtkButton *)
02502     gtk_button_new_with_mnemonic (_("Simulator program"));
02503 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02504                             _("Simulator program executable file"));
02505 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02506 g_signal_connect (window->button_simulator, "clicked",
02507                   G_CALLBACK (dialog_simulator), NULL);
02508
02509 // Creating the evaluator program label and entry
02510 window->check_evaluator = (GtkCheckButton *)
02511     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02512 g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
02513 window->button_evaluator = (GtkButton *)
02514     gtk_button_new_with_mnemonic (_("Evaluator program"));
02515 gtk_widget_set_tooltip_text
02516     (GTK_WIDGET (window->button_evaluator),
02517      _("Optional evaluator program executable file"));
02518 g_signal_connect (window->button_evaluator, "clicked",
02519                   G_CALLBACK (dialog_evaluator), NULL);
02520
02521 // Creating the cleaner program label and entry
02522 window->check_cleaner = (GtkCheckButton *)
02523     gtk_check_button_new_with_mnemonic (_("Cleaner program"));
02524 g_signal_connect (window->check_cleaner, "toggled", window_update, NULL);
02525 window->button_cleaner = (GtkButton *)
02526     gtk_button_new_with_mnemonic (_("Cleaner program"));
02527 gtk_widget_set_tooltip_text
02528     (GTK_WIDGET (window->button_cleaner),
02529      _("Optional cleaner program executable file"));
02530 g_signal_connect (window->button_cleaner, "clicked",
02531                   G_CALLBACK (dialog_cleaner), NULL);
02532
02533 // Creating the results files labels and entries
02534 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02535 window->entry_result = (GtkEntry *) gtk_entry_new ();
02536 gtk_widget_set_tooltip_text
02537     (GTK_WIDGET (window->entry_result), _("Best results file"));
02538 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02539 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02540 gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02542
02543 // Creating the files grid and attaching widgets
02544 window->grid_files = (GtkGrid *) gtk_grid_new ();
02545 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
02546                 0, 0, 1, 1);
02547 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
02548                 1, 0, 1, 1);
02549 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
02550                 0, 1, 1, 1);
02551 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
02552                 1, 1, 1, 1);
02553 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_cleaner),

```

```

02554         0, 2, 1, 1);
02555     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_cleaner),
02556         1, 2, 1, 1);
02557     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
02558         0, 3, 1, 1);
02559     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
02560         1, 3, 1, 1);
02561     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
02562         0, 4, 1, 1);
02563     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
02564         1, 4, 1, 1);
02565
02566     // Creating the algorithm properties
02567     window->label_simulations = (GtkLabel *) gtk_label_new
02568         (_("Simulations number"));
02569     window->spin_simulations
02570         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02571     gtk_widget_set_tooltip_text
02572         (GTK_WIDGET (window->spin_simulations),
02573         _("Number of simulations to perform for each iteration"));
02574     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02575     window->label_iterations = (GtkLabel *)
02576         gtk_label_new (_("Iterations number"));
02577     window->spin_iterations
02578         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02579     gtk_widget_set_tooltip_text
02580         (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02581     g_signal_connect
02582         (window->spin_iterations, "value-changed", window_update, NULL);
02583     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02584     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02585     window->spin_tolerance =
02586         (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02587     gtk_widget_set_tooltip_text
02588         (GTK_WIDGET (window->spin_tolerance),
02589         _("Tolerance to set the variable interval on the next iteration"));
02590     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02591     window->spin_bests
02592         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02593     gtk_widget_set_tooltip_text
02594         (GTK_WIDGET (window->spin_bests),
02595         _("Number of best simulations used to set the variable interval "
02596         "on the next iteration"));
02597     window->label_population
02598         = (GtkLabel *) gtk_label_new (_("Population number"));
02599     window->spin_population
02600         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02601     gtk_widget_set_tooltip_text
02602         (GTK_WIDGET (window->spin_population),
02603         _("Number of population for the genetic algorithm"));
02604     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02605     window->label_generations
02606         = (GtkLabel *) gtk_label_new (_("Generations number"));
02607     window->spin_generations
02608         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02609     gtk_widget_set_tooltip_text
02610         (GTK_WIDGET (window->spin_generations),
02611         _("Number of generations for the genetic algorithm"));
02612     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02613     window->spin_mutation
02614         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02615     gtk_widget_set_tooltip_text
02616         (GTK_WIDGET (window->spin_mutation),
02617         _("Ratio of mutation for the genetic algorithm"));
02618     window->label_reproduction
02619         = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02620     window->spin_reproduction
02621         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02622     gtk_widget_set_tooltip_text
02623         (GTK_WIDGET (window->spin_reproduction),
02624         _("Ratio of reproduction for the genetic algorithm"));
02625     window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02626     window->spin_adaptation
02627         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02628     gtk_widget_set_tooltip_text
02629         (GTK_WIDGET (window->spin_adaptation),
02630         _("Ratio of adaptation for the genetic algorithm"));
02631     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02632     window->spin_threshold = (GtkSpinButton *)
02633         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02634         precision[DEFAULT_PRECISION]);
02635     gtk_widget_set_tooltip_text
02636         (GTK_WIDGET (window->spin_threshold),
02637         _("Threshold in the objective function to finish the simulations"));
02638     window->scrolled_threshold = (GtkScrolledWindow *)
02639     #if !GTK4
02640         gtk_scrolled_window_new (NULL, NULL);

```

```

02641 #else
02642     gtk_scrolled_window_new ();
02643 #endif
02644     gtk_scrolled_window_set_child (window->scrolled_threshold,
02645                                     GTK_WIDGET (window->spin_threshold));
02646 //     gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02647 //     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02648 //                             GTK_ALIGN_FILL);
02649
02650 // Creating the hill climbing method properties
02651     window->check_climbing = (GtkCheckButton *)
02652         gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02653     g_signal_connect (window->check_climbing, "toggled", window_update, NULL);
02654     window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02655 #if !GTK4
02656     window->button_climbing[0] = (GtkRadioButton *)
02657         gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02658 #else
02659     window->button_climbing[0] = (GtkCheckButton *)
02660         gtk_check_button_new_with_mnemonic (label_climbing[0]);
02661 #endif
02662     gtk_grid_attach (window->grid_climbing,
02663                     GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02664     g_signal_connect (window->button_climbing[0], "toggled", window_update, NULL);
02665     for (i = 0; ++i < NCLIMBINGS;)
02666     {
02667 #if !GTK4
02668         window->button_climbing[i] = (GtkRadioButton *)
02669             gtk_radio_button_new_with_mnemonic
02670                 (gtk_radio_button_get_group (window->button_climbing[0]),
02671                  label_climbing[i]);
02672 #else
02673         window->button_climbing[i] = (GtkCheckButton *)
02674             gtk_check_button_new_with_mnemonic (label_climbing[i]);
02675         gtk_check_button_set_group (window->button_climbing[i],
02676                                     window->button_climbing[0]);
02677 #endif
02678         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02679                                     tip_climbing[i]);
02680         gtk_grid_attach (window->grid_climbing,
02681                         GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02682         g_signal_connect (window->button_climbing[i], "toggled", window_update,
02683                         NULL);
02684     }
02685     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02686     window->spin_steps = (GtkSpinButton *)
02687         gtk_spin_button_new_with_range (1., 1.e12, 1.);
02688     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02689     window->label_final_steps
02690         = (GtkLabel *) gtk_label_new (_("Final steps number"));
02691     window->spin_final_steps = (GtkSpinButton *)
02692         gtk_spin_button_new_with_range (1., 1.e12, 1.);
02693     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_final_steps), TRUE);
02694     window->label_estimates
02695         = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02696     window->spin_estimates = (GtkSpinButton *)
02697         gtk_spin_button_new_with_range (1., 1.e3, 1.);
02698     window->label_relaxation
02699         = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02700     window->spin_relaxation = (GtkSpinButton *)
02701         gtk_spin_button_new_with_range (0., 2., 0.001);
02702     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
02703                     0, NCLIMBINGS, 1, 1);
02704     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
02705                     1, NCLIMBINGS, 1, 1);
02706     gtk_grid_attach (window->grid_climbing,
02707                     GTK_WIDGET (window->label_final_steps),
02708                     0, NCLIMBINGS + 1, 1, 1);
02709     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_final_steps),
02710                     1, NCLIMBINGS + 1, 1, 1);
02711     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
02712                     0, NCLIMBINGS + 2, 1, 1);
02713     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
02714                     1, NCLIMBINGS + 2, 1, 1);
02715     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
02716                     0, NCLIMBINGS + 3, 1, 1);
02717     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
02718                     1, NCLIMBINGS + 3, 1, 1);
02719
02720 // Creating the array of algorithms
02721     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02722 #if !GTK4
02723     window->button_algorithm[0] = (GtkRadioButton *)
02724         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02725 #else
02726     window->button_algorithm[0] = (GtkCheckButton *)
02727         gtk_check_button_new_with_mnemonic (label_algorithm[0]);

```



```

02728 #endif
02729 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02730                             tip_algorithm[0]);
02731 gtk_grid_attach (window->grid_algorithm,
02732                 GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02733 g_signal_connect (window->button_algorithm[0], "toggled",
02734                  window_set_algorithm, NULL);
02735 for (i = 0; ++i < NALGORITHMS;)
02736 {
02737 #if !GTK4
02738     window->button_algorithm[i] = (GtkRadioButton *)
02739     gtk_radio_button_new_with_mnemonic
02740     (gtk_radio_button_get_group (window->button_algorithm[0]),
02741      label_algorithm[i]);
02742 #else
02743     window->button_algorithm[i] = (GtkCheckButton *)
02744     gtk_check_button_new_with_mnemonic (label_algorithm[i]);
02745     gtk_check_button_set_group (window->button_algorithm[i],
02746                                window->button_algorithm[0]);
02747 #endif
02748     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02749                                 tip_algorithm[i]);
02750     gtk_grid_attach (window->grid_algorithm,
02751                     GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02752     g_signal_connect (window->button_algorithm[i], "toggled",
02753                      window_set_algorithm, NULL);
02754 }
02755 gtk_grid_attach (window->grid_algorithm,
02756                 GTK_WIDGET (window->label_simulations),
02757                 0, NALGORITHMS, 1, 1);
02758 gtk_grid_attach (window->grid_algorithm,
02759                 GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02760 gtk_grid_attach (window->grid_algorithm,
02761                 GTK_WIDGET (window->label_iterations),
02762                 0, NALGORITHMS + 1, 1, 1);
02763 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
02764                 1, NALGORITHMS + 1, 1, 1);
02765 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
02766                 0, NALGORITHMS + 2, 1, 1);
02767 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
02768                 1, NALGORITHMS + 2, 1, 1);
02769 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
02770                 0, NALGORITHMS + 3, 1, 1);
02771 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
02772                 1, NALGORITHMS + 3, 1, 1);
02773 gtk_grid_attach (window->grid_algorithm,
02774                 GTK_WIDGET (window->label_population),
02775                 0, NALGORITHMS + 4, 1, 1);
02776 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
02777                 1, NALGORITHMS + 4, 1, 1);
02778 gtk_grid_attach (window->grid_algorithm,
02779                 GTK_WIDGET (window->label_generations),
02780                 0, NALGORITHMS + 5, 1, 1);
02781 gtk_grid_attach (window->grid_algorithm,
02782                 GTK_WIDGET (window->spin_generations),
02783                 1, NALGORITHMS + 5, 1, 1);
02784 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
02785                 0, NALGORITHMS + 6, 1, 1);
02786 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
02787                 1, NALGORITHMS + 6, 1, 1);
02788 gtk_grid_attach (window->grid_algorithm,
02789                 GTK_WIDGET (window->label_reproduction),
02790                 0, NALGORITHMS + 7, 1, 1);
02791 gtk_grid_attach (window->grid_algorithm,
02792                 GTK_WIDGET (window->spin_reproduction),
02793                 1, NALGORITHMS + 7, 1, 1);
02794 gtk_grid_attach (window->grid_algorithm,
02795                 GTK_WIDGET (window->label_adaptation),
02796                 0, NALGORITHMS + 8, 1, 1);
02797 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),
02798                 1, NALGORITHMS + 8, 1, 1);
02799 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
02800                 0, NALGORITHMS + 9, 2, 1);
02801 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
02802                 0, NALGORITHMS + 10, 2, 1);
02803 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
02804                 0, NALGORITHMS + 11, 1, 1);
02805 gtk_grid_attach (window->grid_algorithm,
02806                 GTK_WIDGET (window->scrolled_threshold),
02807                 1, NALGORITHMS + 11, 1, 1);
02808 window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02809 gtk_frame_set_child (window->frame_algorithm,
02810                    GTK_WIDGET (window->grid_algorithm));
02811
02812 // Creating the variable widgets
02813 window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02814 gtk_widget_set_tooltip_text

```



```

02815     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02816     window->id_variable = g_signal_connect
02817     (window->combo_variable, "changed", window_set_variable, NULL);
02818 #if !GTK4
02819     window->button_add_variable = (GtkButton *)
02820     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02821 #else
02822     window->button_add_variable = (GtkButton *)
02823     gtk_button_new_from_icon_name ("list-add");
02824 #endif
02825     g_signal_connect (window->button_add_variable, "clicked", window_add_variable,
02826     NULL);
02827     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02828     _("Add variable"));
02829 #if !GTK4
02830     window->button_remove_variable = (GtkButton *)
02831     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02832 #else
02833     window->button_remove_variable = (GtkButton *)
02834     gtk_button_new_from_icon_name ("list-remove");
02835 #endif
02836     g_signal_connect (window->button_remove_variable, "clicked",
02837     window_remove_variable, NULL);
02838     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02839     _("Remove variable"));
02840     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02841     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02842     gtk_widget_set_tooltip_text
02843     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02844     gtk_widget_set_hexexpand (GTK_WIDGET (window->entry_variable), TRUE);
02845     window->id_variable_label = g_signal_connect
02846     (window->entry_variable, "changed", window_label_variable, NULL);
02847     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02848     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02849     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02850     gtk_widget_set_tooltip_text
02851     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02852     window->scrolled_min = (GtkScrolledWindow *)
02853     #if !GTK4
02854     gtk_scrolled_window_new (NULL, NULL);
02855     #else
02856     gtk_scrolled_window_new ();
02857     #endif
02858     gtk_scrolled_window_set_child (window->scrolled_min,
02859     GTK_WIDGET (window->spin_min));
02860     g_signal_connect (window->spin_min, "value-changed",
02861     window_rangemin_variable, NULL);
02862     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02863     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02864     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02865     gtk_widget_set_tooltip_text
02866     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02867     window->scrolled_max = (GtkScrolledWindow *)
02868     #if !GTK4
02869     gtk_scrolled_window_new (NULL, NULL);
02870     #else
02871     gtk_scrolled_window_new ();
02872     #endif
02873     gtk_scrolled_window_set_child (window->scrolled_max,
02874     GTK_WIDGET (window->spin_max));
02875     g_signal_connect (window->spin_max, "value-changed",
02876     window_rangemax_variable, NULL);
02877     window->check_minabs = (GtkCheckButton *)
02878     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02879     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02880     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02881     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02882     gtk_widget_set_tooltip_text
02883     (GTK_WIDGET (window->spin_minabs),
02884     _("Minimum allowed value of the variable"));
02885     window->scrolled_minabs = (GtkScrolledWindow *)
02886     #if !GTK4
02887     gtk_scrolled_window_new (NULL, NULL);
02888     #else
02889     gtk_scrolled_window_new ();
02890     #endif
02891     gtk_scrolled_window_set_child (window->scrolled_minabs,
02892     GTK_WIDGET (window->spin_minabs));
02893     g_signal_connect (window->spin_minabs, "value-changed",
02894     window_rangeminabs_variable, NULL);
02895     window->check_maxabs = (GtkCheckButton *)
02896     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02897     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02898     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02899     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02900     gtk_widget_set_tooltip_text
02901     (GTK_WIDGET (window->spin_maxabs),

```

```

02902     _("Maximum allowed value of the variable"));
02903     window->scrolled_maxabs = (GtkScrolledWindow *)
02904     #if !GTK4
02905         gtk_scrolled_window_new (NULL, NULL);
02906     #else
02907         gtk_scrolled_window_new ();
02908     #endif
02909     gtk_scrolled_window_set_child (window->scrolled_maxabs,
02910                                   GTK_WIDGET (window->spin_maxabs));
02911     g_signal_connect (window->spin_maxabs, "value-changed",
02912                       window_rangemaxabs_variable, NULL);
02913     window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02914     window->spin_precision = (GtkSpinButton *)
02915         gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02916     gtk_widget_set_tooltip_text
02917         (GTK_WIDGET (window->spin_precision),
02918          _("Number of precision floating point digits\n"
02919            "0 is for integer numbers"));
02920     g_signal_connect (window->spin_precision, "value-changed",
02921                       window_precision_variable, NULL);
02922     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02923     window->spin_sweeps =
02924         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02925     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02926                                 _("Number of steps sweeping the variable"));
02927     g_signal_connect (window->spin_sweeps, "value-changed",
02928                       window_update_variable, NULL);
02929     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02930     window->spin_bits
02931         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02932     gtk_widget_set_tooltip_text
02933         (GTK_WIDGET (window->spin_bits),
02934          _("Number of bits to encode the variable"));
02935     g_signal_connect
02936         (window->spin_bits, "value-changed", window_update_variable, NULL);
02937     window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02938     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02939         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02940     gtk_widget_set_tooltip_text
02941         (GTK_WIDGET (window->spin_step),
02942          _("Initial step size for the hill climbing method"));
02943     window->scrolled_step = (GtkScrolledWindow *)
02944     #if !GTK4
02945         gtk_scrolled_window_new (NULL, NULL);
02946     #else
02947         gtk_scrolled_window_new ();
02948     #endif
02949     gtk_scrolled_window_set_child (window->scrolled_step,
02950                                   GTK_WIDGET (window->spin_step));
02951     g_signal_connect
02952         (window->spin_step, "value-changed", window_step_variable, NULL);
02953     window->grid_variable = (GtkGrid *) gtk_grid_new ();
02954     gtk_grid_attach (window->grid_variable,
02955                     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02956     gtk_grid_attach (window->grid_variable,
02957                     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02958     gtk_grid_attach (window->grid_variable,
02959                     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02960     gtk_grid_attach (window->grid_variable,
02961                     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02962     gtk_grid_attach (window->grid_variable,
02963                     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02964     gtk_grid_attach (window->grid_variable,
02965                     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02966     gtk_grid_attach (window->grid_variable,
02967                     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02968     gtk_grid_attach (window->grid_variable,
02969                     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02970     gtk_grid_attach (window->grid_variable,
02971                     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02972     gtk_grid_attach (window->grid_variable,
02973                     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02974     gtk_grid_attach (window->grid_variable,
02975                     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02976     gtk_grid_attach (window->grid_variable,
02977                     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02978     gtk_grid_attach (window->grid_variable,
02979                     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02980     gtk_grid_attach (window->grid_variable,
02981                     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02982     gtk_grid_attach (window->grid_variable,
02983                     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02984     gtk_grid_attach (window->grid_variable,
02985                     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02986     gtk_grid_attach (window->grid_variable,
02987                     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02988     gtk_grid_attach (window->grid_variable,

```

```

02989         GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02990     gtk_grid_attach (window->grid_variable,
02991         GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02992     gtk_grid_attach (window->grid_variable,
02993         GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02994     gtk_grid_attach (window->grid_variable,
02995         GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02996     window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02997     gtk_frame_set_child (window->frame_variable,
02998         GTK_WIDGET (window->grid_variable));
02999
03000     // Creating the experiment widgets
03001     window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03002     gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03003         _("Experiment selector"));
03004     window->id_experiment = g_signal_connect
03005         (window->combo_experiment, "changed", window_set_experiment, NULL);
03006 #if !GTK4
03007     window->button_add_experiment = (GtkButton *)
03008         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
03009 #else
03010     window->button_add_experiment = (GtkButton *)
03011         gtk_button_new_from_icon_name ("list-add");
03012 #endif
03013     g_signal_connect
03014         (window->button_add_experiment, "clicked", window_add_experiment, NULL);
03015     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03016         _("Add experiment"));
03017 #if !GTK4
03018     window->button_remove_experiment = (GtkButton *)
03019         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
03020 #else
03021     window->button_remove_experiment = (GtkButton *)
03022         gtk_button_new_from_icon_name ("list-remove");
03023 #endif
03024     g_signal_connect (window->button_remove_experiment, "clicked",
03025         window_remove_experiment, NULL);
03026     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03027         _("Remove experiment"));
03028     window->label_experiment
03029         = (GtkLabel *) gtk_label_new (_("Experimental data file"));
03030     window->button_experiment = (GtkButton *)
03031         gtk_button_new_with_mnemonic (_("Experimental data file"));
03032     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03033         _("Experimental data file"));
03034     g_signal_connect (window->button_experiment, "clicked",
03035         window_name_experiment, NULL);
03036     gtk_widget_set_expand (GTK_WIDGET (window->button_experiment), TRUE);
03037     window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
03038     window->spin_weight
03039         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03040     gtk_widget_set_tooltip_text
03041         (GTK_WIDGET (window->spin_weight),
03042         _("Weight factor to build the objective function"));
03043     g_signal_connect
03044         (window->spin_weight, "value-changed", window_weight_experiment, NULL);
03045     window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03046     gtk_grid_attach (window->grid_experiment,
03047         GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03048     gtk_grid_attach (window->grid_experiment,
03049         GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03050     gtk_grid_attach (window->grid_experiment,
03051         GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03052     gtk_grid_attach (window->grid_experiment,
03053         GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03054     gtk_grid_attach (window->grid_experiment,
03055         GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03056     gtk_grid_attach (window->grid_experiment,
03057         GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03058     gtk_grid_attach (window->grid_experiment,
03059         GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03060     for (i = 0; i < MAX_NINPUTS; ++i)
03061     {
03062         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
03063         window->check_template[i] = (GtkCheckButton *)
03064             gtk_check_button_new_with_label (buffer3);
03065         window->id_template[i]
03066             = g_signal_connect (window->check_template[i], "toggled",
03067                 window_inputs_experiment, NULL);
03068         gtk_grid_attach (window->grid_experiment,
03069             GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03070         window->button_template[i] = (GtkButton *)
03071             gtk_button_new_with_mnemonic (_("Input template"));
03072         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
03073             _("Experimental input template file"));
03074         window->id_input[i] =

```

```

03076         g_signal_connect_swapped (window->button_template[i], "clicked",
03077                                     (GCallback) window_template_experiment,
03078                                     (void *) (size_t) i);
03079         gtk_grid_attach (window->grid_experiment,
03080                         GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03081     }
03082     window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
03083     gtk_frame_set_child (window->frame_experiment,
03084                         GTK_WIDGET (window->grid_experiment));
03085
03086     // Creating the error norm widgets
03087     window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
03088     window->grid_norm = (GtkGrid *) gtk_grid_new ();
03089     gtk_frame_set_child (window->frame_norm, GTK_WIDGET (window->grid_norm));
03090     #if !GTK4
03091     window->button_norm[0] = (GtkRadioButton *)
03092         gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
03093     #else
03094     window->button_norm[0] = (GtkCheckButton *)
03095         gtk_check_button_new_with_mnemonic (label_norm[0]);
03096     #endif
03097     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
03098                                 tip_norm[0]);
03099     gtk_grid_attach (window->grid_norm,
03100                     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
03101     g_signal_connect (window->button_norm[0], "toggled", window_update, NULL);
03102     for (i = 0; ++i < NNORMS;)
03103     {
03104     #if !GTK4
03105         window->button_norm[i] = (GtkRadioButton *)
03106             gtk_radio_button_new_with_mnemonic
03107             (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
03108     #else
03109         window->button_norm[i] = (GtkCheckButton *)
03110             gtk_check_button_new_with_mnemonic (label_norm[i]);
03111         gtk_check_button_set_group (window->button_norm[i],
03112                                   window->button_norm[0]);
03113     #endif
03114     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
03115                                 tip_norm[i]);
03116     gtk_grid_attach (window->grid_norm,
03117                     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
03118     g_signal_connect (window->button_norm[i], "toggled", window_update, NULL);
03119     }
03120     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
03121     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
03122     window->spin_p = (GtkSpinButton *)
03123         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
03124     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
03125                                 _("P parameter for the P error norm"));
03126     window->scrolled_p = (GtkScrolledWindow *)
03127         #if !GTK4
03128         gtk_scrolled_window_new (NULL, NULL);
03129     #else
03130         gtk_scrolled_window_new ();
03131     #endif
03132     gtk_scrolled_window_set_child (window->scrolled_p,
03133                                   GTK_WIDGET (window->spin_p));
03134     gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
03135     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
03136     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
03137                     1, 2, 1, 2);
03138
03139     // Creating the grid and attaching the widgets to the grid
03140     window->grid = (GtkGrid *) gtk_grid_new ();
03141     gtk_grid_attach (window->grid, GTK_WIDGET (window->box_buttons), 0, 0, 3, 1);
03142     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
03143     gtk_grid_attach (window->grid,
03144                     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
03145     gtk_grid_attach (window->grid,
03146                     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
03147     gtk_grid_attach (window->grid,
03148                     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
03149     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
03150     gtk_window_set_child (window->window, GTK_WIDGET (window->grid));
03151
03152     // Setting the window logo
03153     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03154     #if !GTK4
03155     gtk_window_set_icon (window->window, window->logo);
03156     #endif
03157
03158     // Showing the window
03159     #if !GTK4
03160     gtk_widget_show_all (GTK_WIDGET (window->window));
03161     #else
03162     gtk_widget_show (GTK_WIDGET (window->window));

```

```

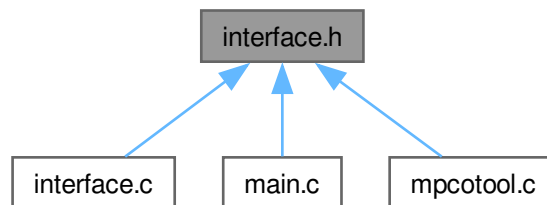
03163 #endif
03164
03165 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
03166 #if GTK_MINOR_VERSION >= 16
03167   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03168   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03169   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03170   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03171   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
03172   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
03173   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
03174 #endif
03175
03176 // Reading initial example
03177 input_new ();
03178 buffer2 = g_get_current_dir ();
03179 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03180 g_free (buffer2);
03181 window_read (buffer);
03182 g_free (buffer);
03183
03184 #if DEBUG_INTERFACE
03185   fprintf (stderr, "window_new: start\n");
03186 #endif
03187 }

```

## 4.13 interface.h File Reference

Header file to define the graphical interface functions.

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



### Data Structures

- struct [Options](#)  
*Struct to define the options dialog.*
- struct [Running](#)  
*Struct to define the running dialog.*
- struct [Window](#)  
*Struct to define the main window.*

### Macros

- #define [MAX\\_LENGTH](#) ([DEFAULT\\_PRECISION](#) + 8)  
*Max length of texts allowed in GtkSpinButtons.*

## Functions

- void [window\\_new](#) (GtkApplication \*application)

## Variables

- [Window window](#) [1]  
*[Window](#) struct to define the main interface window.*

### 4.13.1 Detailed Description

Header file to define the graphical interface functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [interface.h](#).

### 4.13.2 Macro Definition Documentation

#### 4.13.2.1 MAX\_LENGTH

```
#define MAX_LENGTH (DEFAULT_PRECISION + 8)
```

Max length of texts allowed in GtkSpinButtons.

Definition at line [42](#) of file [interface.h](#).

### 4.13.3 Function Documentation

#### 4.13.3.1 window\_new()

```
void window_new (  
    GtkApplication * application )
```

Function to open the main window.

#### Parameters

<i>application</i>	GtkApplication struct.
--------------------	------------------------

Definition at line 2363 of file [interface.c](#).

```

02364 {
02365     unsigned int i;
02366     char *buffer, *buffer2, buffer3[64];
02367     const char *label_algorithm[NALGORITHMS] = {
02368         "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02369     };
02370     const char *tip_algorithm[NALGORITHMS] = {
02371         _("Monte-Carlo brute force algorithm"),
02372         _("Sweep brute force algorithm"),
02373         _("Genetic algorithm"),
02374         _("Orthogonal sampling brute force algorithm"),
02375     };
02376     const char *label_climbing[NCLIMBINGS] = {
02377         _("_Coordinates climbing"), _("_Random climbing")
02378     };
02379     const char *tip_climbing[NCLIMBINGS] = {
02380         _("Coordinates climbing estimate method"),
02381         _("Random climbing estimate method")
02382     };
02383     const char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02384     const char *tip_norm[NNORMS] = {
02385         _("Euclidean error norm (L2)"),
02386         _("Maximum error norm (L)"),
02387         _("P error norm (Lp)"),
02388         _("Taxicab error norm (L1)")
02389     };
02390     #if !GTK4
02391     const char *close = "delete-event";
02392     #else
02393     const char *close = "close-request";
02394     #endif
02395
02396     #if DEBUG_INTERFACE
02397     fprintf (stderr, "window_new: start\n");
02398     #endif
02399
02400     // Creating the window
02401     window->window = window_parent = main_window
02402         = (GtkWindow *) gtk_application_window_new (application);
02403
02404     // Finish when closing the window
02405     g_signal_connect_swapped (window->window, close,
02406                             G_CALLBACK (g_application_quit),
02407                             G_APPLICATION (application));
02408
02409     // Setting the window title
02410     gtk_window_set_title (window->window, "MPCOTool");
02411
02412     // Creating the open button
02413     window->button_open = (GtkButton *)
02414     #if !GTK4
02415         gtk_button_new_from_icon_name ("document-open", GTK_ICON_SIZE_BUTTON);
02416     #else
02417         gtk_button_new_from_icon_name ("document-open");
02418     #endif
02419     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_open),
02420                                 _("Open a case"));
02421     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02422
02423     // Creating the save button
02424     window->button_save = (GtkButton *)
02425     #if !GTK4
02426         gtk_button_new_from_icon_name ("document-save", GTK_ICON_SIZE_BUTTON);
02427     #else
02428         gtk_button_new_from_icon_name ("document-save");
02429     #endif
02430     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_save),
02431                                 _("Save the case"));
02432     g_signal_connect (window->button_save, "clicked", (GCallback) window_save,
02433                     NULL);
02434
02435     // Creating the run button
02436     window->button_run = (GtkButton *)
02437     #if !GTK4
02438         gtk_button_new_from_icon_name ("system-run", GTK_ICON_SIZE_BUTTON);
02439     #else
02440         gtk_button_new_from_icon_name ("system-run");
02441     #endif
02442     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_run),
02443                                 _("Run the optimization"));
02444     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02445
02446     // Creating the options button
02447     window->button_options = (GtkButton *)
02448     #if !GTK4
02449         gtk_button_new_from_icon_name ("preferences-system", GTK_ICON_SIZE_BUTTON);

```

```

02450 #else
02451     gtk_button_new_from_icon_name ("preferences-system");
02452 #endif
02453     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_options),
02454     _("Edit the case"));
02455     g_signal_connect (window->button_options, "clicked", options_new, NULL);
02456
02457     // Creating the help button
02458     window->button_help = (GtkButton *)
02459 #if !GTK4
02460     gtk_button_new_from_icon_name ("help-browser", GTK_ICON_SIZE_BUTTON);
02461 #else
02462     gtk_button_new_from_icon_name ("help-browser");
02463 #endif
02464     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_help), _("Help"));
02465     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02466
02467     // Creating the about button
02468     window->button_about = (GtkButton *)
02469 #if !GTK4
02470     gtk_button_new_from_icon_name ("help-about", GTK_ICON_SIZE_BUTTON);
02471 #else
02472     gtk_button_new_from_icon_name ("help-about");
02473 #endif
02474     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_about), _("About"));
02475     g_signal_connect (window->button_about, "clicked", window_about, NULL);
02476
02477     // Creating the exit button
02478     window->button_exit = (GtkButton *)
02479 #if !GTK4
02480     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
02481 #else
02482     gtk_button_new_from_icon_name ("application-exit");
02483 #endif
02484     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_exit), _("Exit"));
02485     g_signal_connect_swapped (window->button_exit, "clicked",
02486     G_CALLBACK (g_application_quit),
02487     G_APPLICATION (application));
02488
02489     // Creating the buttons bar
02490     window->box_buttons = (GtkBox *) gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
02491     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_open));
02492     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_save));
02493     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_run));
02494     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_options));
02495     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_help));
02496     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_about));
02497     gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_exit));
02498
02499     // Creating the simulator program label and entry
02500     window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02501     window->button_simulator = (GtkButton *)
02502     gtk_button_new_with_mnemonic (_("Simulator program"));
02503     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02504     _("Simulator program executable file"));
02505     gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02506     g_signal_connect (window->button_simulator, "clicked",
02507     G_CALLBACK (dialog_simulator), NULL);
02508
02509     // Creating the evaluator program label and entry
02510     window->check_evaluator = (GtkCheckButton *)
02511     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02512     g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
02513     window->button_evaluator = (GtkButton *)
02514     gtk_button_new_with_mnemonic (_("Evaluator program"));
02515     gtk_widget_set_tooltip_text
02516     (GTK_WIDGET (window->button_evaluator),
02517     _("Optional evaluator program executable file"));
02518     g_signal_connect (window->button_evaluator, "clicked",
02519     G_CALLBACK (dialog_evaluator), NULL);
02520
02521     // Creating the cleaner program label and entry
02522     window->check_cleaner = (GtkCheckButton *)
02523     gtk_check_button_new_with_mnemonic (_("_Cleaner program"));
02524     g_signal_connect (window->check_cleaner, "toggled", window_update, NULL);
02525     window->button_cleaner = (GtkButton *)
02526     gtk_button_new_with_mnemonic (_("Cleaner program"));
02527     gtk_widget_set_tooltip_text
02528     (GTK_WIDGET (window->button_cleaner),
02529     _("Optional cleaner program executable file"));
02530     g_signal_connect (window->button_cleaner, "clicked",
02531     G_CALLBACK (dialog_cleaner), NULL);
02532
02533     // Creating the results files labels and entries
02534     window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02535     window->entry_result = (GtkEntry *) gtk_entry_new ();
02536     gtk_widget_set_tooltip_text

```



```
02537     (GTK_WIDGET (window->entry_result), _("Best results file"));
02538     window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02539     window->entry_variables = (GtkEntry *) gtk_entry_new ();
02540     gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02542
02543     // Creating the files grid and attaching widgets
02544     window->grid_files = (GtkGrid *) gtk_grid_new ();
02545     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
02546                     0, 0, 1, 1);
02547     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
02548                     1, 0, 1, 1);
02549     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
02550                     0, 1, 1, 1);
02551     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
02552                     1, 1, 1, 1);
02553     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_cleaner),
02554                     0, 2, 1, 1);
02555     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_cleaner),
02556                     1, 2, 1, 1);
02557     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
02558                     0, 3, 1, 1);
02559     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
02560                     1, 3, 1, 1);
02561     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
02562                     0, 4, 1, 1);
02563     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
02564                     1, 4, 1, 1);
02565
02566     // Creating the algorithm properties
02567     window->label_simulations = (GtkLabel *) gtk_label_new
02568     (_("Simulations number"));
02569     window->spin_simulations
02570     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02571     gtk_widget_set_tooltip_text
02572     (GTK_WIDGET (window->spin_simulations),
02573      _("Number of simulations to perform for each iteration"));
02574     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02575     window->label_iterations = (GtkLabel *)
02576     gtk_label_new (_("Iterations number"));
02577     window->spin_iterations
02578     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02579     gtk_widget_set_tooltip_text
02580     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02581     g_signal_connect
02582     (window->spin_iterations, "value-changed", window_update, NULL);
02583     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02584     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02585     window->spin_tolerance =
02586     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02587     gtk_widget_set_tooltip_text
02588     (GTK_WIDGET (window->spin_tolerance),
02589      _("Tolerance to set the variable interval on the next iteration"));
02590     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02591     window->spin_bests
02592     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02593     gtk_widget_set_tooltip_text
02594     (GTK_WIDGET (window->spin_bests),
02595      _("Number of best simulations used to set the variable interval "
02596        "on the next iteration"));
02597     window->label_population
02598     = (GtkLabel *) gtk_label_new (_("Population number"));
02599     window->spin_population
02600     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02601     gtk_widget_set_tooltip_text
02602     (GTK_WIDGET (window->spin_population),
02603      _("Number of population for the genetic algorithm"));
02604     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02605     window->label_generations
02606     = (GtkLabel *) gtk_label_new (_("Generations number"));
02607     window->spin_generations
02608     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02609     gtk_widget_set_tooltip_text
02610     (GTK_WIDGET (window->spin_generations),
02611      _("Number of generations for the genetic algorithm"));
02612     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02613     window->spin_mutation
02614     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02615     gtk_widget_set_tooltip_text
02616     (GTK_WIDGET (window->spin_mutation),
02617      _("Ratio of mutation for the genetic algorithm"));
02618     window->label_reproduction
02619     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02620     window->spin_reproduction
02621     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02622     gtk_widget_set_tooltip_text
02623     (GTK_WIDGET (window->spin_reproduction),
```

```

02624     _("Ratio of reproduction for the genetic algorithm"));
02625 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02626 window->spin_adaptation
02627 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02628 gtk_widget_set_tooltip_text
02629 (GTK_WIDGET (window->spin_adaptation),
02630  _("Ratio of adaptation for the genetic algorithm"));
02631 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02632 window->spin_threshold = (GtkSpinButton *)
02633   gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02634   precision[DEFAULT_PRECISION]);
02635 gtk_widget_set_tooltip_text
02636 (GTK_WIDGET (window->spin_threshold),
02637  _("Threshold in the objective function to finish the simulations"));
02638 window->scrolled_threshold = (GtkScrolledWindow *)
02639 #if !GTK4
02640   gtk_scrolled_window_new (NULL, NULL);
02641 #else
02642   gtk_scrolled_window_new ();
02643 #endif
02644 gtk_scrolled_window_set_child (window->scrolled_threshold,
02645   GTK_WIDGET (window->spin_threshold));
02646 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02647 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02648 //   GTK_ALIGN_FILL);
02649 // Creating the hill climbing method properties
02650 window->check_climbing = (GtkCheckButton *)
02651   gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02652 g_signal_connect (window->check_climbing, "toggled", window_update, NULL);
02653 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02654 #if !GTK4
02655 window->button_climbing[0] = (GtkRadioButton *)
02656   gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02657 #else
02658 window->button_climbing[0] = (GtkCheckButton *)
02659   gtk_check_button_new_with_mnemonic (label_climbing[0]);
02660 #endif
02661 gtk_grid_attach (window->grid_climbing,
02662   GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02663 g_signal_connect (window->button_climbing[0], "toggled", window_update, NULL);
02664 for (i = 0; ++i < NCLIMBINGS;)
02665 {
02666 #if !GTK4
02667   window->button_climbing[i] = (GtkRadioButton *)
02668     gtk_radio_button_new_with_mnemonic
02669       (gtk_radio_button_get_group (window->button_climbing[0]),
02670        label_climbing[i]);
02671 #else
02672   window->button_climbing[i] = (GtkCheckButton *)
02673     gtk_check_button_new_with_mnemonic (label_climbing[i]);
02674   gtk_check_button_set_group (window->button_climbing[i],
02675     window->button_climbing[0]);
02676 #endif
02677   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02678     tip_climbing[i]);
02679   gtk_grid_attach (window->grid_climbing,
02680     GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02681   g_signal_connect (window->button_climbing[i], "toggled", window_update,
02682     NULL);
02683 }
02684 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02685 window->spin_steps = (GtkSpinButton *)
02686   gtk_spin_button_new_with_range (1., 1.e12, 1.);
02687 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02688 window->label_final_steps
02689 = (GtkLabel *) gtk_label_new (_("Final steps number"));
02690 window->spin_final_steps = (GtkSpinButton *)
02691   gtk_spin_button_new_with_range (1., 1.e12, 1.);
02692 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_final_steps), TRUE);
02693 window->label_estimates
02694 = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02695 window->spin_estimates = (GtkSpinButton *)
02696   gtk_spin_button_new_with_range (1., 1.e3, 1.);
02697 window->label_relaxation
02698 = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02699 window->spin_relaxation = (GtkSpinButton *)
02700   gtk_spin_button_new_with_range (0., 2., 0.001);
02701 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
02702   0, NCLIMBINGS, 1, 1);
02703 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
02704   1, NCLIMBINGS, 1, 1);
02705 gtk_grid_attach (window->grid_climbing,
02706   GTK_WIDGET (window->label_final_steps),
02707   0, NCLIMBINGS + 1, 1, 1);
02708 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_final_steps),
02709   1, NCLIMBINGS + 1, 1, 1);
02710

```

```

02711     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
02712                     0, NCLIMBINGS + 2, 1, 1);
02713     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
02714                     1, NCLIMBINGS + 2, 1, 1);
02715     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
02716                     0, NCLIMBINGS + 3, 1, 1);
02717     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
02718                     1, NCLIMBINGS + 3, 1, 1);
02719
02720     // Creating the array of algorithms
02721     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02722     #if !GTK4
02723     window->button_algorithm[0] = (GtkRadioButton *)
02724     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02725     #else
02726     window->button_algorithm[0] = (GtkCheckButton *)
02727     gtk_check_button_new_with_mnemonic (label_algorithm[0]);
02728     #endif
02729     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02730                                 tip_algorithm[0]);
02731     gtk_grid_attach (window->grid_algorithm,
02732                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02733     g_signal_connect (window->button_algorithm[0], "toggled",
02734                     window_set_algorithm, NULL);
02735     for (i = 0; ++i < NALGORITHMS;)
02736     {
02737     #if !GTK4
02738     window->button_algorithm[i] = (GtkRadioButton *)
02739     gtk_radio_button_new_with_mnemonic
02740     (gtk_radio_button_get_group (window->button_algorithm[0]),
02741     label_algorithm[i]);
02742     #else
02743     window->button_algorithm[i] = (GtkCheckButton *)
02744     gtk_check_button_new_with_mnemonic (label_algorithm[i]);
02745     gtk_check_button_set_group (window->button_algorithm[i],
02746     window->button_algorithm[0]);
02747     #endif
02748     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02749     tip_algorithm[i]);
02750     gtk_grid_attach (window->grid_algorithm,
02751                     GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02752     g_signal_connect (window->button_algorithm[i], "toggled",
02753     window_set_algorithm, NULL);
02754     }
02755     gtk_grid_attach (window->grid_algorithm,
02756                     GTK_WIDGET (window->label_simulations),
02757                     0, NALGORITHMS, 1, 1);
02758     gtk_grid_attach (window->grid_algorithm,
02759                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02760     gtk_grid_attach (window->grid_algorithm,
02761                     GTK_WIDGET (window->label_iterations),
02762                     0, NALGORITHMS + 1, 1, 1);
02763     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
02764                     1, NALGORITHMS + 1, 1, 1);
02765     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
02766                     0, NALGORITHMS + 2, 1, 1);
02767     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
02768                     1, NALGORITHMS + 2, 1, 1);
02769     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
02770                     0, NALGORITHMS + 3, 1, 1);
02771     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
02772                     1, NALGORITHMS + 3, 1, 1);
02773     gtk_grid_attach (window->grid_algorithm,
02774                     GTK_WIDGET (window->label_population),
02775                     0, NALGORITHMS + 4, 1, 1);
02776     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
02777                     1, NALGORITHMS + 4, 1, 1);
02778     gtk_grid_attach (window->grid_algorithm,
02779                     GTK_WIDGET (window->label_generations),
02780                     0, NALGORITHMS + 5, 1, 1);
02781     gtk_grid_attach (window->grid_algorithm,
02782                     GTK_WIDGET (window->spin_generations),
02783                     1, NALGORITHMS + 5, 1, 1);
02784     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
02785                     0, NALGORITHMS + 6, 1, 1);
02786     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
02787                     1, NALGORITHMS + 6, 1, 1);
02788     gtk_grid_attach (window->grid_algorithm,
02789                     GTK_WIDGET (window->label_reproduction),
02790                     0, NALGORITHMS + 7, 1, 1);
02791     gtk_grid_attach (window->grid_algorithm,
02792                     GTK_WIDGET (window->spin_reproduction),
02793                     1, NALGORITHMS + 7, 1, 1);
02794     gtk_grid_attach (window->grid_algorithm,
02795                     GTK_WIDGET (window->label_adaptation),
02796                     0, NALGORITHMS + 8, 1, 1);
02797     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),

```

```

02798         1, NALGORITHMS + 8, 1, 1);
02799     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
02800         0, NALGORITHMS + 9, 2, 1);
02801     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
02802         0, NALGORITHMS + 10, 2, 1);
02803     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
02804         0, NALGORITHMS + 11, 1, 1);
02805     gtk_grid_attach (window->grid_algorithm,
02806         GTK_WIDGET (window->scrolled_threshold),
02807         1, NALGORITHMS + 11, 1, 1);
02808     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02809     gtk_frame_set_child (window->frame_algorithm,
02810         GTK_WIDGET (window->grid_algorithm));
02811
02812     // Creating the variable widgets
02813     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02814     gtk_widget_set_tooltip_text
02815         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02816     window->id_variable = g_signal_connect
02817         (window->combo_variable, "changed", window_set_variable, NULL);
02818     #if !GTK4
02819     window->button_add_variable = (GtkButton *)
02820         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02821     #else
02822     window->button_add_variable = (GtkButton *)
02823         gtk_button_new_from_icon_name ("list-add");
02824     #endif
02825     g_signal_connect (window->button_add_variable, "clicked", window_add_variable,
02826         NULL);
02827     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02828         _("Add variable"));
02829     #if !GTK4
02830     window->button_remove_variable = (GtkButton *)
02831         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02832     #else
02833     window->button_remove_variable = (GtkButton *)
02834         gtk_button_new_from_icon_name ("list-remove");
02835     #endif
02836     g_signal_connect (window->button_remove_variable, "clicked",
02837         window_remove_variable, NULL);
02838     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02839         _("Remove variable"));
02840     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02841     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02842     gtk_widget_set_tooltip_text
02843         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02844     gtk_widget_set_hexexpand (GTK_WIDGET (window->entry_variable), TRUE);
02845     window->id_variable_label = g_signal_connect
02846         (window->entry_variable, "changed", window_label_variable, NULL);
02847     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02848     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02849         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02850     gtk_widget_set_tooltip_text
02851         (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02852     window->scrolled_min = (GtkScrolledWindow *)
02853     #if !GTK4
02854         gtk_scrolled_window_new (NULL, NULL);
02855     #else
02856         gtk_scrolled_window_new ();
02857     #endif
02858     gtk_scrolled_window_set_child (window->scrolled_min,
02859         GTK_WIDGET (window->spin_min));
02860     g_signal_connect (window->spin_min, "value-changed",
02861         window_rangemin_variable, NULL);
02862     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02863     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02864         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02865     gtk_widget_set_tooltip_text
02866         (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02867     window->scrolled_max = (GtkScrolledWindow *)
02868     #if !GTK4
02869         gtk_scrolled_window_new (NULL, NULL);
02870     #else
02871         gtk_scrolled_window_new ();
02872     #endif
02873     gtk_scrolled_window_set_child (window->scrolled_max,
02874         GTK_WIDGET (window->spin_max));
02875     g_signal_connect (window->spin_max, "value-changed",
02876         window_rangemax_variable, NULL);
02877     window->check_minabs = (GtkCheckButton *)
02878         gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02879     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02880     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02881         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02882     gtk_widget_set_tooltip_text
02883         (GTK_WIDGET (window->spin_minabs),
02884         _("Minimum allowed value of the variable"));

```

```

02885     window->scrolled_minabs = (GtkScrolledWindow *)
02886     #if !GTK4
02887         gtk_scrolled_window_new (NULL, NULL);
02888     #else
02889         gtk_scrolled_window_new ();
02890     #endif
02891     gtk_scrolled_window_set_child (window->scrolled_minabs,
02892                                   GTK_WIDGET (window->spin_minabs));
02893     g_signal_connect (window->spin_minabs, "value-changed",
02894                       window_rangeminabs_variable, NULL);
02895     window->check_maxabs = (GtkCheckButton *)
02896     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02897     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02898     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02899     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02900     gtk_widget_set_tooltip_text
02901     (GTK_WIDGET (window->spin_maxabs),
02902      _("Maximum allowed value of the variable"));
02903     window->scrolled_maxabs = (GtkScrolledWindow *)
02904     #if !GTK4
02905         gtk_scrolled_window_new (NULL, NULL);
02906     #else
02907         gtk_scrolled_window_new ();
02908     #endif
02909     gtk_scrolled_window_set_child (window->scrolled_maxabs,
02910                                   GTK_WIDGET (window->spin_maxabs));
02911     g_signal_connect (window->spin_maxabs, "value-changed",
02912                       window_rangemaxabs_variable, NULL);
02913     window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02914     window->spin_precision = (GtkSpinButton *)
02915     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02916     gtk_widget_set_tooltip_text
02917     (GTK_WIDGET (window->spin_precision),
02918      _("Number of precision floating point digits\n"
02919        "0 is for integer numbers"));
02920     g_signal_connect (window->spin_precision, "value-changed",
02921                       window_precision_variable, NULL);
02922     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02923     window->spin_sweeps =
02924     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02925     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02926      _("Number of steps sweeping the variable"));
02927     g_signal_connect (window->spin_sweeps, "value-changed",
02928                       window_update_variable, NULL);
02929     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02930     window->spin_bits
02931     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02932     gtk_widget_set_tooltip_text
02933     (GTK_WIDGET (window->spin_bits),
02934      _("Number of bits to encode the variable"));
02935     g_signal_connect
02936     (window->spin_bits, "value-changed", window_update_variable, NULL);
02937     window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02938     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02939     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02940     gtk_widget_set_tooltip_text
02941     (GTK_WIDGET (window->spin_step),
02942      _("Initial step size for the hill climbing method"));
02943     window->scrolled_step = (GtkScrolledWindow *)
02944     #if !GTK4
02945         gtk_scrolled_window_new (NULL, NULL);
02946     #else
02947         gtk_scrolled_window_new ();
02948     #endif
02949     gtk_scrolled_window_set_child (window->scrolled_step,
02950                                   GTK_WIDGET (window->spin_step));
02951     g_signal_connect
02952     (window->spin_step, "value-changed", window_step_variable, NULL);
02953     window->grid_variable = (GtkGrid *) gtk_grid_new ();
02954     gtk_grid_attach (window->grid_variable,
02955                     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02956     gtk_grid_attach (window->grid_variable,
02957                     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02958     gtk_grid_attach (window->grid_variable,
02959                     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02960     gtk_grid_attach (window->grid_variable,
02961                     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02962     gtk_grid_attach (window->grid_variable,
02963                     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02964     gtk_grid_attach (window->grid_variable,
02965                     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02966     gtk_grid_attach (window->grid_variable,
02967                     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02968     gtk_grid_attach (window->grid_variable,
02969                     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02970     gtk_grid_attach (window->grid_variable,
02971                     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);

```

```

02972 gtk_grid_attach (window->grid_variable,
02973                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02974 gtk_grid_attach (window->grid_variable,
02975                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02976 gtk_grid_attach (window->grid_variable,
02977                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02978 gtk_grid_attach (window->grid_variable,
02979                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02980 gtk_grid_attach (window->grid_variable,
02981                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02982 gtk_grid_attach (window->grid_variable,
02983                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02984 gtk_grid_attach (window->grid_variable,
02985                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02986 gtk_grid_attach (window->grid_variable,
02987                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02988 gtk_grid_attach (window->grid_variable,
02989                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02990 gtk_grid_attach (window->grid_variable,
02991                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02992 gtk_grid_attach (window->grid_variable,
02993                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02994 gtk_grid_attach (window->grid_variable,
02995                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02996 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02997 gtk_frame_set_child (window->frame_variable,
02998                    GTK_WIDGET (window->grid_variable));
02999
03000 // Creating the experiment widgets
03001 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03002 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03003                             _("Experiment selector"));
03004 window->id_experiment = g_signal_connect
03005     (window->combo_experiment, "changed", window_set_experiment, NULL);
03006 #if !GTK4
03007 window->button_add_experiment = (GtkButton *)
03008     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
03009 #else
03010 window->button_add_experiment = (GtkButton *)
03011     gtk_button_new_from_icon_name ("list-add");
03012 #endif
03013 g_signal_connect
03014     (window->button_add_experiment, "clicked", window_add_experiment, NULL);
03015 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03016                             _("Add experiment"));
03017 #if !GTK4
03018 window->button_remove_experiment = (GtkButton *)
03019     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
03020 #else
03021 window->button_remove_experiment = (GtkButton *)
03022     gtk_button_new_from_icon_name ("list-remove");
03023 #endif
03024 g_signal_connect (window->button_remove_experiment, "clicked",
03025                 window_remove_experiment, NULL);
03026 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03027                             _("Remove experiment"));
03028 window->label_experiment
03029     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
03030 window->button_experiment = (GtkButton *)
03031     gtk_button_new_with_mnemonic (_("Experimental data file"));
03032 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03033                             _("Experimental data file"));
03034 g_signal_connect (window->button_experiment, "clicked",
03035                 window_name_experiment, NULL);
03036 gtk_widget_set_expand (GTK_WIDGET (window->button_experiment), TRUE);
03037 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
03038 window->spin_weight
03039     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03040 gtk_widget_set_tooltip_text
03041     (GTK_WIDGET (window->spin_weight),
03042      _("Weight factor to build the objective function"));
03043 g_signal_connect
03044     (window->spin_weight, "value-changed", window_weight_experiment, NULL);
03045 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03046 gtk_grid_attach (window->grid_experiment,
03047                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03048 gtk_grid_attach (window->grid_experiment,
03049                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03050 gtk_grid_attach (window->grid_experiment,
03051                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03052 gtk_grid_attach (window->grid_experiment,
03053                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03054 gtk_grid_attach (window->grid_experiment,
03055                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03056 gtk_grid_attach (window->grid_experiment,
03057                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03058 gtk_grid_attach (window->grid_experiment,

```



```

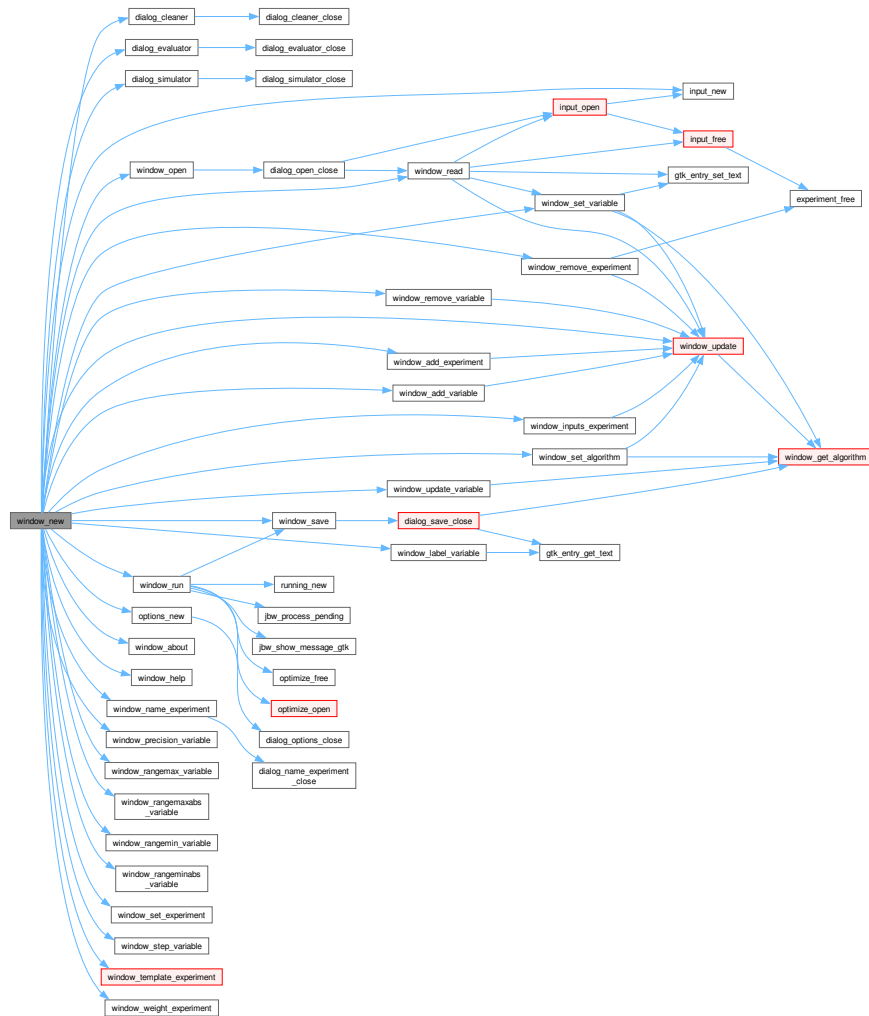
03059         GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03060     for (i = 0; i < MAX_NINPUTS; ++i)
03061     {
03062         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
03063         window->check_template[i] = (GtkCheckButton *)
03064             gtk_check_button_new_with_label (buffer3);
03065         window->id_template[i]
03066             = g_signal_connect (window->check_template[i], "toggled",
03067                 window_inputs_experiment, NULL);
03068         gtk_grid_attach (window->grid_experiment,
03069             GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03070         window->button_template[i] = (GtkButton *)
03071             gtk_button_new_with_mnemonic (_("Input template"));
03072
03073         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
03074             _("Experimental input template file"));
03075         window->id_input[i] =
03076             g_signal_connect_swapped (window->button_template[i], "clicked",
03077                 (GCallback) window_template_experiment,
03078                 (void *) (size_t) i);
03079         gtk_grid_attach (window->grid_experiment,
03080             GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03081     }
03082     window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
03083     gtk_frame_set_child (window->frame_experiment,
03084         GTK_WIDGET (window->grid_experiment));
03085
03086     // Creating the error norm widgets
03087     window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
03088     window->grid_norm = (GtkGrid *) gtk_grid_new ();
03089     gtk_frame_set_child (window->frame_norm, GTK_WIDGET (window->grid_norm));
03090     #if !GTK4
03091     window->button_norm[0] = (GtkRadioButton *)
03092         gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
03093     #else
03094     window->button_norm[0] = (GtkCheckButton *)
03095         gtk_check_button_new_with_mnemonic (label_norm[0]);
03096     #endif
03097     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
03098         tip_norm[0]);
03099     gtk_grid_attach (window->grid_norm,
03100         GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
03101     g_signal_connect (window->button_norm[0], "toggled", window_update, NULL);
03102     for (i = 0; ++i < NNORMS;)
03103     {
03104     #if !GTK4
03105         window->button_norm[i] = (GtkRadioButton *)
03106             gtk_radio_button_new_with_mnemonic
03107                 (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
03108     #else
03109         window->button_norm[i] = (GtkCheckButton *)
03110             gtk_check_button_new_with_mnemonic (label_norm[i]);
03111         gtk_check_button_set_group (window->button_norm[i],
03112             window->button_norm[0]);
03113     #endif
03114         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
03115             tip_norm[i]);
03116         gtk_grid_attach (window->grid_norm,
03117             GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
03118         g_signal_connect (window->button_norm[i], "toggled", window_update, NULL);
03119     }
03120     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
03121     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
03122     window->spin_p = (GtkSpinButton *)
03123         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
03124     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
03125         _("P parameter for the P error norm"));
03126     window->scrolled_p = (GtkScrolledWindow *)
03127     #if !GTK4
03128         gtk_scrolled_window_new (NULL, NULL);
03129     #else
03130         gtk_scrolled_window_new ();
03131     #endif
03132     gtk_scrolled_window_set_child (window->scrolled_p,
03133         GTK_WIDGET (window->spin_p));
03134     gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
03135     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
03136     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
03137         1, 2, 1, 2);
03138
03139     // Creating the grid and attaching the widgets to the grid
03140     window->grid = (GtkGrid *) gtk_grid_new ();
03141     gtk_grid_attach (window->grid, GTK_WIDGET (window->box_buttons), 0, 0, 3, 1);
03142     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
03143     gtk_grid_attach (window->grid,
03144         GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
03145     gtk_grid_attach (window->grid,

```

```
03146         GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
03147     gtk_grid_attach (window->grid,
03148         GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
03149     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
03150     gtk_window_set_child (window->window, GTK_WIDGET (window->grid));
03151
03152     // Setting the window logo
03153     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03154     #if !GTK4
03155     gtk_window_set_icon (window->window, window->logo);
03156     #endif
03157
03158     // Showing the window
03159     #if !GTK4
03160     gtk_widget_show_all (GTK_WIDGET (window->window));
03161     #else
03162     gtk_widget_show (GTK_WIDGET (window->window));
03163     #endif
03164
03165     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
03166     #if GTK_MINOR_VERSION >= 16
03167     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03168     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03169     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03170     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03171     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
03172     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
03173     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
03174     #endif
03175
03176     // Reading initial example
03177     input_new ();
03178     buffer2 = g_get_current_dir ();
03179     buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03180     g_free (buffer2);
03181     window_read (buffer);
03182     g_free (buffer);
03183
03184     #if DEBUG_INTERFACE
03185     fprintf (stderr, "window_new: start\n");
03186     #endif
03187 }
```



Here is the call graph for this function:



## 4.13.4 Variable Documentation

### 4.13.4.1 window

`Window window[1] [extern]`

`Window` struct to define the main interface window.

Definition at line 81 of file [interface.c](#).

## 4.14 interface.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
```

```

00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS" AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INTERFACE__H
00033 #define INTERFACE__H 1
00034
00035 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00036
00037 typedef struct
00038 {
00039     GtkWidget *dialog;
00040     GtkWidget *grid;
00041     GtkWidget *label_seed;
00042     GtkWidget *spin_seed;
00043     GtkWidget *label_threads;
00044     GtkWidget *spin_threads;
00045     GtkWidget *label_climbing;
00046     GtkWidget *spin_climbing;
00047 } Options;
00048
00049 typedef struct
00050 {
00051     GtkWidget *dialog;
00052     GtkWidget *label;
00053     GtkWidget *spinner;
00054     GtkWidget *grid;
00055 } Running;
00056
00057 typedef struct
00058 {
00059     GtkWidget *window;
00060     GtkWidget *grid;
00061     GtkWidget *box_buttons;
00062     GtkWidget *button_open;
00063     GtkWidget *button_save;
00064     GtkWidget *button_run;
00065     GtkWidget *button_options;
00066     GtkWidget *button_help;
00067     GtkWidget *button_about;
00068     GtkWidget *button_exit;
00069     GtkWidget *grid_files;
00070     GtkWidget *label_simulator;
00071     GtkWidget *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *button_evaluator;
00074     GtkWidget *check_cleaner;
00075     GtkWidget *button_cleaner;
00076     GtkWidget *label_result;
00077     GtkWidget *entry_result;
00078     GtkWidget *label_variables;
00079     GtkWidget *entry_variables;
00080     GtkWidget *frame_norm;
00081     GtkWidget *grid_norm;
00082 #if !GTK4
00083     GtkWidget *button_norm[NNORMS];
00084 #else
00085     GtkWidget *button_norm[NNORMS];
00086 #endif
00087     GtkWidget *label_p;
00088     GtkWidget *spin_p;
00089     GtkWidget *scrolled_p;
00090     GtkWidget *frame_algorithm;
00091     GtkWidget *grid_algorithm;

```

```

00116 #if !GTK4
00117     GtkRadioButton *button_algorithm[NALGORITHMS];
00119 #else
00120     GtkCheckButton *button_algorithm[NALGORITHMS];
00122 #endif
00123     GtkLabel *label_simulations;
00124     GtkSpinButton *spin_simulations;
00126     GtkLabel *label_iterations;
00127     GtkSpinButton *spin_iterations;
00129     GtkLabel *label_tolerance;
00130     GtkSpinButton *spin_tolerance;
00131     GtkLabel *label_bests;
00132     GtkSpinButton *spin_bests;
00133     GtkLabel *label_population;
00134     GtkSpinButton *spin_population;
00136     GtkLabel *label_generations;
00137     GtkSpinButton *spin_generations;
00139     GtkLabel *label_mutation;
00140     GtkSpinButton *spin_mutation;
00141     GtkLabel *label_reproduction;
00142     GtkSpinButton *spin_reproduction;
00144     GtkLabel *label_adaptation;
00145     GtkSpinButton *spin_adaptation;
00147     GtkCheckButton *check_climbing;
00149     GtkGrid *grid_climbing;
00151 #if !GTK4
00152     GtkRadioButton *button_climbing[NCLIMBINGS];
00154 #else
00155     GtkCheckButton *button_climbing[NCLIMBINGS];
00157 #endif
00158     GtkLabel *label_steps;
00159     GtkSpinButton *spin_steps;
00160     GtkLabel *label_final_steps;
00161     GtkSpinButton *spin_final_steps;
00163     GtkLabel *label_estimates;
00164     GtkSpinButton *spin_estimates;
00166     GtkLabel *label_relaxation;
00168     GtkSpinButton *spin_relaxation;
00170     GtkLabel *label_threshold;
00171     GtkSpinButton *spin_threshold;
00172     GtkScrolledWindow *scrolled_threshold;
00174     GtkFrame *frame_variable;
00175     GtkGrid *grid_variable;
00176     GtkComboBoxText *combo_variable;
00178     GtkButton *button_add_variable;
00179     GtkButton *button_remove_variable;
00180     GtkLabel *label_variable;
00181     GtkEntry *entry_variable;
00182     GtkLabel *label_min;
00183     GtkSpinButton *spin_min;
00184     GtkScrolledWindow *scrolled_min;
00185     GtkLabel *label_max;
00186     GtkSpinButton *spin_max;
00187     GtkScrolledWindow *scrolled_max;
00188     GtkCheckButton *check_minabs;
00189     GtkSpinButton *spin_minabs;
00190     GtkScrolledWindow *scrolled_minabs;
00191     GtkCheckButton *check_maxabs;
00192     GtkSpinButton *spin_maxabs;
00193     GtkScrolledWindow *scrolled_maxabs;
00194     GtkLabel *label_precision;
00195     GtkSpinButton *spin_precision;
00196     GtkLabel *label_sweeps;
00197     GtkSpinButton *spin_sweeps;
00198     GtkLabel *label_bits;
00199     GtkSpinButton *spin_bits;
00200     GtkLabel *label_step;
00201     GtkSpinButton *spin_step;
00202     GtkScrolledWindow *scrolled_step;
00203     GtkFrame *frame_experiment;
00204     GtkGrid *grid_experiment;
00205     GtkComboBoxText *combo_experiment;
00206     GtkButton *button_add_experiment;
00207     GtkButton *button_remove_experiment;
00208     GtkLabel *label_experiment;
00209     GtkButton *button_experiment;
00211     GtkLabel *label_weight;
00212     GtkSpinButton *spin_weight;
00213     GtkCheckButton *check_template[MAX_NINPUTS];
00215     GtkButton *button_template[MAX_NINPUTS];
00217     GdkPixbuf *logo;
00218     Experiment *experiment;
00219     Variable *variable;
00220     char *application_directory;
00221     gulong id_experiment;
00222     gulong id_experiment_name;
00223     gulong id_variable;

```

```

00224     gulong id_variable_label;
00225     gulong id_template[MAX_NINPUTS];
00227     gulong id_input[MAX_NINPUTS];
00229     unsigned int nexperiments;
00230     unsigned int nvariables;
00231 } Window;
00232
00233 // Global variables
00234 extern Window window[1];
00235
00236 // Public functions
00237 void window_new (GtkApplication * application);
00238
00239 #endif

```

## 4.15 main.c File Reference

Main source file.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "jb/src/jb_win.h"
#include "genetic/genetic.h"
#include "tools.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
#include "mpcotool.h"

```

Include dependency graph for main.c:



### Macros

- `#define JBW 2`

### Functions

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

### 4.15.1 Detailed Description

Main source file.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [main.c](#).

### 4.15.2 Macro Definition Documentation

#### 4.15.2.1 JBW

```
#define JBW 2
```

Definition at line 59 of file [main.c](#).

### 4.15.3 Function Documentation

#### 4.15.3.1 main()

```
int main (  
    int argn,  
    char ** argc )
```

Main function

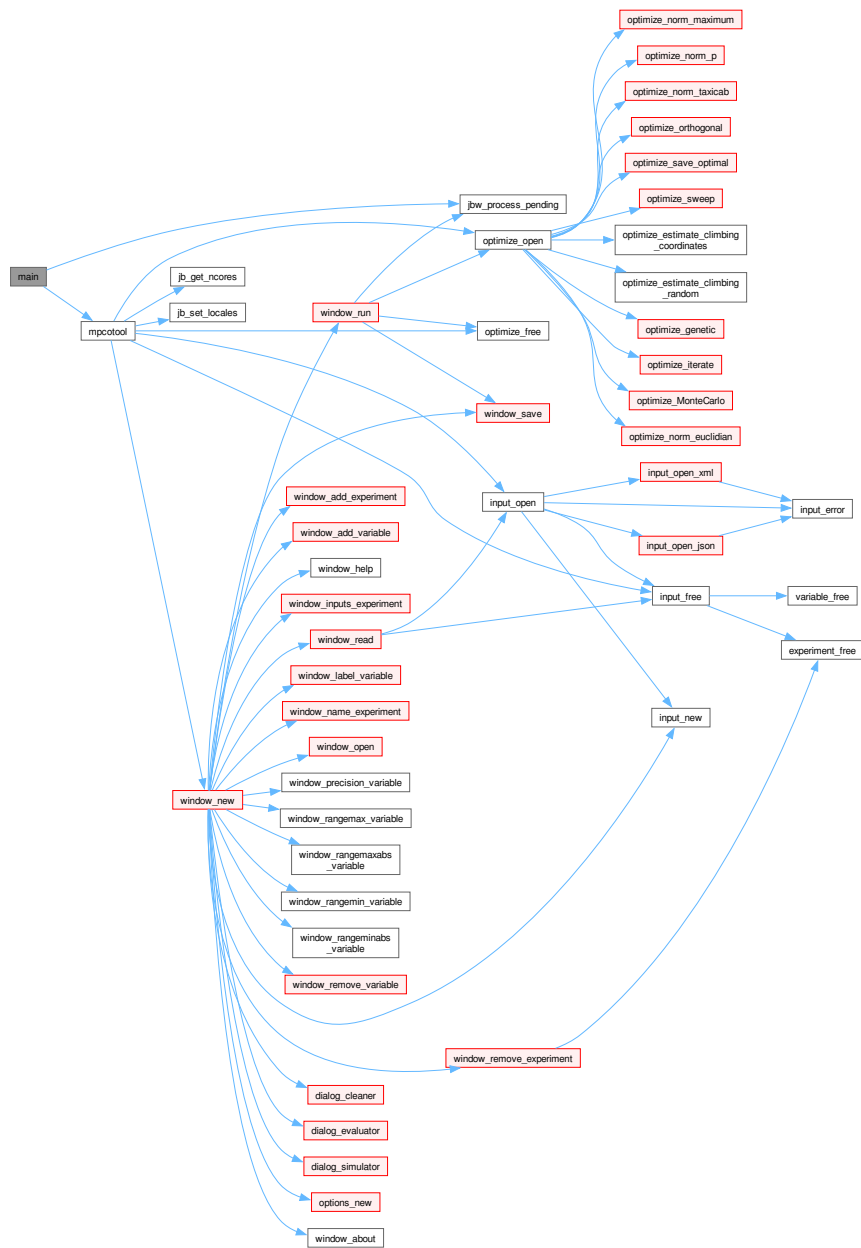
#### Returns

0 on succes, error code (>0) on error.

Definition at line 81 of file [main.c](#).

```
00082 {  
00083     #if HAVE_GTK  
00084         show_pending = jbw_process_pending;  
00085     #endif  
00086     jbw_init (&argn, &argc);  
00087     return mpcotool (argn, argc);  
00088 }
```

Here is the call graph for this function:



## 4.16 main.c

[Go to the documentation of this file.](#)

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
  
```

```

00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #if HAVE_GTK
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
00053 #define JBW 2
00054 #else
00055 #define JBW 1
00056 #endif
00057 #include "jb/src/jb_win.h"
00058 #include "genetic/genetic.h"
00059 #include "tools.h"
00060 #include "experiment.h"
00061 #include "variable.h"
00062 #include "input.h"
00063 #include "optimize.h"
00064 #if HAVE_GTK
00065 #include "interface.h"
00066 #endif
00067 #include "mpcotool.h"
00068
00069 int
00070 main (int argn, char **argc)
00071 {
00072     #if HAVE_GTK
00073         show_pending = jbw_process_pending;
00074     #endif
00075     jbw_init (&argn, &argc);
00076     return mpcotool (argn, argc);
00077 }

```

## 4.17 mpcotool.c File Reference

Main function source file.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <math.h>

```

```

#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "jb/src/jb_win.h"
#include "genetic/genetic.h"
#include "tools.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
#include "mpcotool.h"

```

Include dependency graph for mpcotool.c:



## Macros

- `#define DEBUG\_MPCOTOOL 1`  
Macro to debug main functions.

## Functions

- `int mpcotool (int argn, char **argc)`

### 4.17.1 Detailed Description

Main function source file.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [mpcotool.c](#).



## 4.17.2 Macro Definition Documentation

### 4.17.2.1 DEBUG\_MPCOTOOL

```
#define DEBUG_MPCOTOOL 1
```

Macro to debug main functions.

Definition at line 73 of file [mpcotool.c](#).

## 4.17.3 Function Documentation

### 4.17.3.1 mpcotool()

```
int mpcotool (
    int argn,
    char ** argc )
```

Main function.

#### Returns

0 on success, >0 on error.

#### Parameters

<i>argn</i>	Arguments number.
<i>argc</i>	Arguments pointer.

Definition at line 81 of file [mpcotool.c](#).

```
00083 {
00084     const struct option options[] = {
00085         {"seed", required_argument, NULL, 's'},
00086         {"nthreads", required_argument, NULL, 't'},
00087         {NULL, 0, NULL, 0}
00088     };
00089     #if HAVE_GTK
00090     GtkApplication *application;
00091     #endif
00092     int o, option_index;
00093
00094     // Starting pseudo-random numbers generator
00095     #if DEBUG_MPCOTOOL
00096     fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00097     #endif
00098     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00099
00100     // Allowing spaces in the XML data file
00101     #if DEBUG_MPCOTOOL
00102     fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00103     #endif
00104     xmlKeepBlanksDefault (0);
00105
00106     // Starting MPI
00107     #if HAVE_MPI
00108     #if DEBUG_MPCOTOOL
00109     fprintf (stderr, "mpcotool: starting MPI\n");
00110     #endif
00111     MPI_Init (&argn, &argc);
00112     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00113     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00114     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00115     #else
```

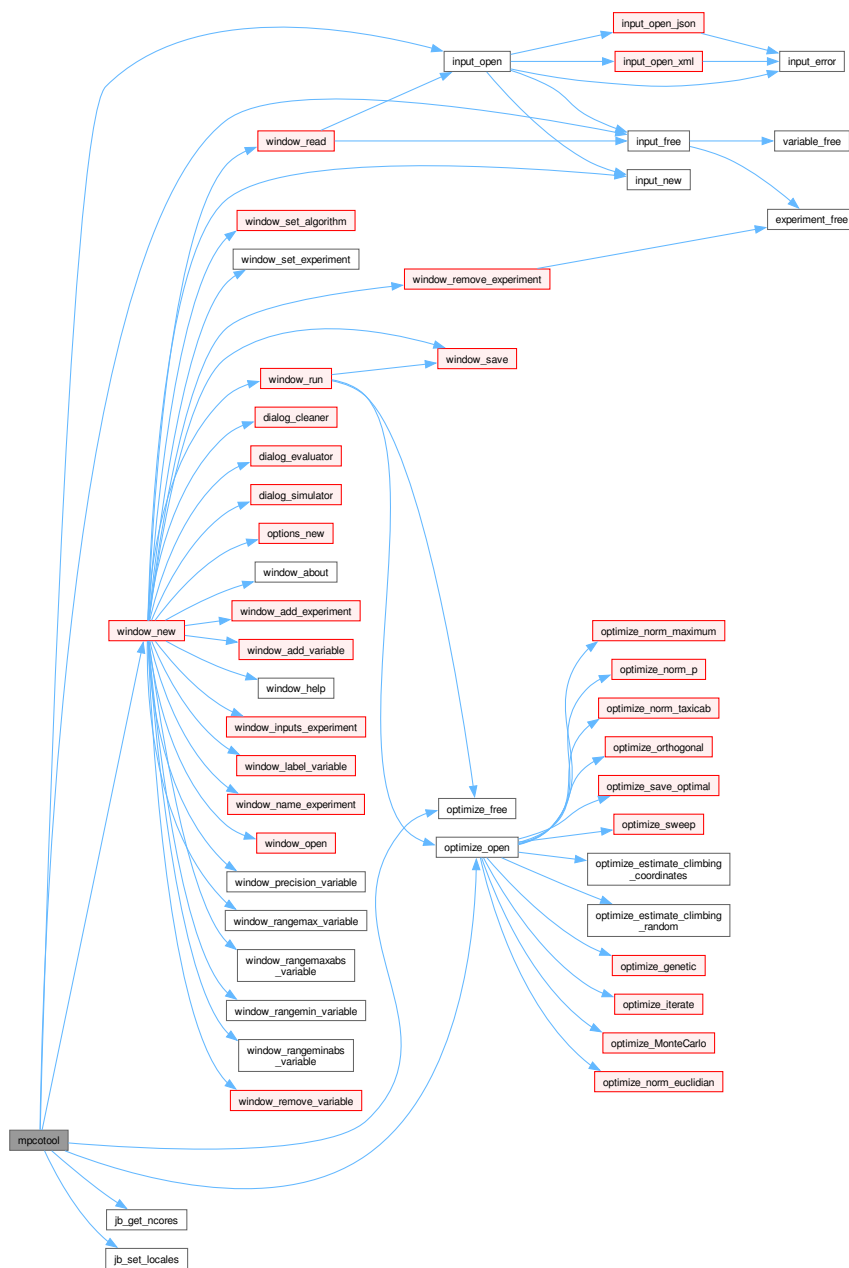
```

00116     ntasks = 1;
00117 #endif
00118
00119 // Getting threads number and pseudo-random numbers generator seed
00120 nthreads_climbing = nthreads = jb_get_ncores ();
00121 optimize->seed = DEFAULT_RANDOM_SEED;
00122
00123 // Parsing command line arguments
00124 while (1)
00125 {
00126     o = getopt_long (argn, argc, "s:t:", options, &option_index);
00127     if (o == -1)
00128         break;
00129     switch (o)
00130     {
00131         case 's':
00132             optimize->seed = atol (optarg);
00133             break;
00134         case 't':
00135             nthreads_climbing = nthreads = atoi (optarg);
00136             break;
00137         default:
00138             printf ("%s\n%s\n", _("ERROR!"), _("Unknown option"));
00139             return 1;
00140     }
00141 }
00142 argn -= optind;
00143
00144 // Resetting result and variables file names
00145 #if DEBUG_MPCOTOOL
00146 fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00147 #endif
00148 input->result = input->variables = NULL;
00149
00150 #if HAVE_GTK
00151
00152 // Setting local language and international floating point numbers notation
00153 jb_set_locales (PROGRAM_INTERFACE, LOCALE_DIR, "", "C");
00154
00155 // Initing GTK+
00156 window->application_directory = g_get_current_dir ();
00157 gtk_disable_setlocale ();
00158 application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00159                                   G_APPLICATION_DEFAULT_FLAGS);
00160 g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00161
00162 // Opening the main window
00163 g_application_run (G_APPLICATION (application), 0, NULL);
00164
00165 // Freeing memory
00166 input_free ();
00167 gtk_window_destroy (window->window);
00168 g_object_unref (application);
00169 g_free (window->application_directory);
00170
00171 #else
00172
00173 // Checking syntax
00174 if (argn < 1 || argn > 3)
00175 {
00176     printf ("The syntax is:\n"
00177            " ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00178            "[variables_file]\n");
00179     return 2;
00180 }
00181 if (argn > 1)
00182     input->result = (char *) xmlStrdup ((xmlChar *) argc[optind + 1]);
00183 if (argn == 2)
00184     input->variables = (char *) xmlStrdup ((xmlChar *) argc[optind + 2]);
00185
00186 // Making optimization
00187 #if DEBUG_MPCOTOOL
00188 fprintf (stderr, "mpcotool: making optimization\n");
00189 #endif
00190 if (input_open (argc[optind]))
00191     optimize_open ();
00192
00193 // Freeing memory
00194 #if DEBUG_MPCOTOOL
00195 fprintf (stderr, "mpcotool: freeing memory and closing\n");
00196 #endif
00197 optimize_free ();
00198
00199 #endif
00200
00201 // Closing MPI
00202 #if HAVE_MPI

```

```
00203 MPI_Finalize ();
00204 #endif
00205
00206 // Freeing memory
00207 gsl_rng_free (optimize->rng);
00208
00209 // Closing
00210 return 0;
00211 }
```

Here is the call graph for this function:



#### 4.18 mpcotool.c

[Go to the documentation of this file.](#)

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS" AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <getopt.h>
00038 #include <math.h>
00039 #include <locale.h>
00040 #include <gsl/gsl_rng.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #ifdef G_OS_WIN32
00046 #include <windows.h>
00047 #endif
00048 #if HAVE_MPI
00049 #include <mpi.h>
00050 #endif
00051 #if HAVE_GTK
00052 #include <gio/gio.h>
00053 #include <gtk/gtk.h>
00054 #endif
00055 #include "jb/src/jb_win.h"
00056 #include "genetic/genetic.h"
00057 #include "tools.h"
00058 #include "experiment.h"
00059 #include "variable.h"
00060 #include "input.h"
00061 #include "optimize.h"
00062 #if HAVE_GTK
00063 #include "interface.h"
00064 #endif
00065 #include "mpcotool.h"
00066
00067 #define DEBUG_MPCOTOOL 1
00068
00069 int
00070 mpcotool (int argn,
00071          char **argv)
00072 {
00073     const struct option options[] = {
00074         {"seed", required_argument, NULL, 's'},
00075         {"nthreads", required_argument, NULL, 't'},
00076         {NULL, 0, NULL, 0}
00077     };
00078
00079     #if HAVE_GTK
00080     GtkApplication *application;
00081     #endif
00082     int o, option_index;
00083
00084     // Starting pseudo-random numbers generator
00085     #if DEBUG_MPCOTOOL
00086     fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00087     #endif
00088     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);

```

```

00099
00100 // Allowing spaces in the XML data file
00101 #if DEBUG_MPCOTOOL
00102 fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00103 #endif
00104 xmlKeepBlanksDefault (0);
00105
00106 // Starting MPI
00107 #if HAVE_MPI
00108 #if DEBUG_MPCOTOOL
00109 fprintf (stderr, "mpcotool: starting MPI\n");
00110 #endif
00111 MPI_Init (&argn, &argc);
00112 MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00113 MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00114 printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00115 #else
00116 ntasks = 1;
00117 #endif
00118
00119 // Getting threads number and pseudo-random numbers generator seed
00120 nthreads_climbing = nthreads = jb_get_ncores ();
00121 optimize->seed = DEFAULT_RANDOM_SEED;
00122
00123 // Parsing command line arguments
00124 while (1)
00125 {
00126     o = getopt_long (argn, argc, "s:t:", options, &option_index);
00127     if (o == -1)
00128         break;
00129     switch (o)
00130     {
00131         case 's':
00132             optimize->seed = atol (optarg);
00133             break;
00134         case 't':
00135             nthreads_climbing = nthreads = atoi (optarg);
00136             break;
00137         default:
00138             printf ("%s\n%s\n", _("ERROR!"), _("Unknown option"));
00139             return 1;
00140     }
00141 }
00142 argn -= optind;
00143
00144 // Resetting result and variables file names
00145 #if DEBUG_MPCOTOOL
00146 fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00147 #endif
00148 input->result = input->variables = NULL;
00149
00150 #if HAVE_GTK
00151
00152 // Setting local language and international floating point numbers notation
00153 jb_set_locales (PROGRAM_INTERFACE, LOCALE_DIR, "", "C");
00154
00155 // Initing GTK+
00156 window->application_directory = g_get_current_dir ();
00157 gtk_disable_setlocale ();
00158 application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00159                                     G_APPLICATION_DEFAULT_FLAGS);
00160 g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00161
00162 // Opening the main window
00163 g_application_run (G_APPLICATION (application), 0, NULL);
00164
00165 // Freeing memory
00166 input_free ();
00167 gtk_window_destroy (window->window);
00168 g_object_unref (application);
00169 g_free (window->application_directory);
00170
00171 #else
00172
00173 // Checking syntax
00174 if (argn < 1 || argn > 3)
00175 {
00176     printf ("The syntax is:\n"
00177             " ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00178             "[variables_file]\n");
00179     return 2;
00180 }
00181 if (argn > 1)
00182     input->result = (char *) xmlStrdup ((xmlChar *) argc[optind + 1]);
00183 if (argn == 2)
00184     input->variables = (char *) xmlStrdup ((xmlChar *) argc[optind + 2]);
00185

```

```

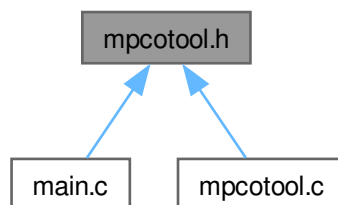
00186 // Making optimization
00187 #if DEBUG_MPCOTOOL
00188     fprintf (stderr, "mpcotool: making optimization\n");
00189 #endif
00190     if (input_open (argc[optind]))
00191         optimize_open ();
00192
00193 // Freeing memory
00194 #if DEBUG_MPCOTOOL
00195     fprintf (stderr, "mpcotool: freeing memory and closing\n");
00196 #endif
00197     optimize_free ();
00198 #endif
00199
00200
00201 // Closing MPI
00202 #if HAVE_MPI
00203     MPI_Finalize ();
00204 #endif
00205
00206 // Freeing memory
00207     gsl_rng_free (optimize->rng);
00208
00209 // Closing
00210     return 0;
00211 }

```

## 4.19 mpcotool.h File Reference

Main function header file.

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



### Functions

- int [mpcotool](#) (int argn, char \*\*argc)

### 4.19.1 Detailed Description

Main function header file.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [mpcotool.h](#).

## 4.19.2 Function Documentation

### 4.19.2.1 mpcotool()

```
int mpcotool (
    int argn,
    char ** argc ) [extern]
```

Main function.

#### Returns

0 on success, >0 on error.

#### Parameters

<i>argn</i>	Arguments number.
<i>argc</i>	Arguments pointer.

Definition at line 81 of file [mpcotool.c](#).

```
00083 {
00084     const struct option options[] = {
00085         {"seed", required_argument, NULL, 's'},
00086         {"nthreads", required_argument, NULL, 't'},
00087         {NULL, 0, NULL, 0}
00088     };
00089     #if HAVE_GTK
00090     GtkApplication *application;
00091     #endif
00092     int o, option_index;
00093
00094     // Starting pseudo-random numbers generator
00095     #if DEBUG_MPCOTOOL
00096     fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00097     #endif
00098     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00099
00100     // Allowing spaces in the XML data file
00101     #if DEBUG_MPCOTOOL
00102     fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00103     #endif
00104     xmlKeepBlanksDefault (0);
00105
00106     // Starting MPI
00107     #if HAVE_MPI
00108     #if DEBUG_MPCOTOOL
00109     fprintf (stderr, "mpcotool: starting MPI\n");
00110     #endif
00111     MPI_Init (&argn, &argc);
00112     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00113     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00114     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00115     #else
00116     ntasks = 1;
00117     #endif
00118
00119     // Getting threads number and pseudo-random numbers generator seed
00120     nthreads_climbing = nthreads = jb_get_ncores ();
00121     optimize->seed = DEFAULT_RANDOM_SEED;
00122
00123     // Parsing command line arguments
00124     while (1)
00125     {
00126         o = getopt_long (argn, argc, "s:t:", options, &option_index);
00127         if (o == -1)
00128             break;
00129         switch (o)
00130         {
00131             case 's':
00132                 optimize->seed = atol (optarg);
00133                 break;
00134             case 't':
```

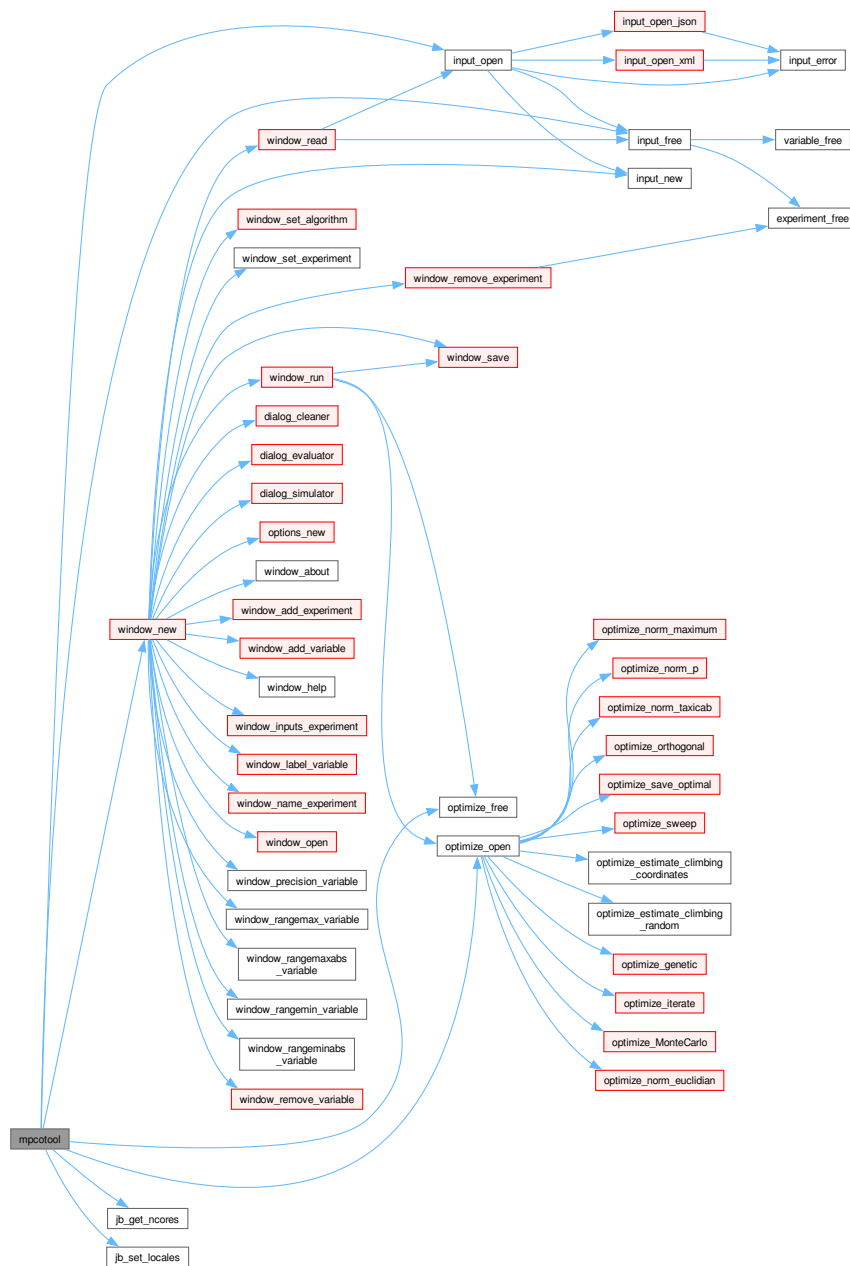
```

00135         nthreads_climbing = nthreads = atoi (optarg);
00136         break;
00137     default:
00138         printf ("%s\n%s\n", _("ERROR!"), _("Unknown option"));
00139         return 1;
00140     }
00141 }
00142 argn -= optind;
00143
00144 // Resetting result and variables file names
00145 #if DEBUG_MPCOTOOL
00146     fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00147 #endif
00148     input->result = input->variables = NULL;
00149
00150 #if HAVE_GTK
00151
00152     // Setting local language and international floating point numbers notation
00153     jb_set_locales (PROGRAM_INTERFACE, LOCALE_DIR, "", "C");
00154
00155     // Initing GTK+
00156     window->application_directory = g_get_current_dir ();
00157     gtk_disable_setlocale ();
00158     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00159                                     G_APPLICATION_DEFAULT_FLAGS);
00160     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00161
00162     // Opening the main window
00163     g_application_run (G_APPLICATION (application), 0, NULL);
00164
00165     // Freeing memory
00166     input_free ();
00167     gtk_window_destroy (window->window);
00168     g_object_unref (application);
00169     g_free (window->application_directory);
00170
00171 #else
00172
00173     // Checking syntax
00174     if (argn < 1 || argn > 3)
00175     {
00176         printf ("The syntax is:\n"
00177                "  ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00178                "[variables_file]\n");
00179         return 2;
00180     }
00181     if (argn > 1)
00182         input->result = (char *) xmlStrdup ((xmlChar *) argc[optind + 1]);
00183     if (argn == 2)
00184         input->variables = (char *) xmlStrdup ((xmlChar *) argc[optind + 2]);
00185
00186     // Making optimization
00187     #if DEBUG_MPCOTOOL
00188         fprintf (stderr, "mpcotool: making optimization\n");
00189     #endif
00190     if (input_open (argc[optind]))
00191         optimize_open ();
00192
00193     // Freeing memory
00194     #if DEBUG_MPCOTOOL
00195         fprintf (stderr, "mpcotool: freeing memory and closing\n");
00196     #endif
00197     optimize_free ();
00198
00199 #endif
00200
00201 // Closing MPI
00202 #if HAVE_MPI
00203     MPI_Finalize ();
00204 #endif
00205
00206 // Freeing memory
00207     gsl_rng_free (optimize->rng);
00208
00209 // Closing
00210     return 0;
00211 }

```



Here is the call graph for this function:



## 4.20 mpcotool.h

[Go to the documentation of this file.](#)

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,

```

```

00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS" AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef MPCOTOOL__H
00033 #define MPCOTOOL__H 1
00034
00035 extern int mpcotool (int argn, char **argc);
00036
00037 #endif

```

## 4.21 optimize.c File Reference

Source file to define the optimization functions.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/param.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <alloca.h>
#include <mpi.h>
#include "jb/src/jb_win.h"
#include "genetic/genetic.h"
#include "tools.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"

```

Include dependency graph for optimize.c:



### Macros

- #define `DEBUG_OPTIMIZE` 0

*Macro to debug optimize functions.*

- `#define CP "cp"`

*Macro to define the shell copy command.*

- `#define RM "rm"`

*Macro to define the shell remove command.*

## Functions

- static void `optimize_input` (unsigned int simulation, char \*input, GMappedFile \*stencil)
- static double `optimize_parse` (unsigned int simulation, unsigned int experiment)
- static double `optimize_norm_euclidian` (unsigned int simulation)
- static double `optimize_norm_maximum` (unsigned int simulation)
- static double `optimize_norm_p` (unsigned int simulation)
- static double `optimize_norm_taxicab` (unsigned int simulation)
- static void `optimize_print` ()
- static void `optimize_save_variables` (unsigned int simulation, double error)
- static void `optimize_best` (unsigned int simulation, double value)
- static void `optimize_sequential` ()
- static void \* `optimize_thread` (ParallelData \*data)
- static void `optimize_merge` (unsigned int nsaveds, unsigned int \*simulation\_best, double \*error\_best)
- static void `optimize_synchronise` ()
- static void `optimize_sweep` ()
- static void `optimize_MonteCarlo` ()
- static void `optimize_orthogonal` ()
- static void `optimize_best_climbing` (unsigned int simulation, double value)
- static void `optimize_climbing_sequential` (unsigned int simulation)
- static void \* `optimize_climbing_thread` (ParallelData \*data)
- static double `optimize_estimate_climbing_random` (unsigned int variable, unsigned int estimate)
- static double `optimize_estimate_climbing_coordinates` (unsigned int variable, unsigned int estimate)
- static void `optimize_step_climbing` (unsigned int simulation)
- static void `optimize_climbing_best` ()
- static void `optimize_climbing` (unsigned int nsteps)
- static double `optimize_genetic_objective` ( Entity \*entity)
- static void `optimize_genetic` ()
- static void `optimize_save_old` ()
- static void `optimize_merge_old` ()
- static void `optimize_refine` ()
- static void `optimize_step` ()
- static void `optimize_iterate` ()
- static void `optimize_save_optimal` ()
- void `optimize_free` ()
- void `optimize_open` ()

## Variables

- `Optimize optimize` [1]  
*Optimization data.*
- unsigned int `nthreads_climbing`  
*Number of threads for the hill climbing method.*
- static void(\* `optimize_algorithm` )()  
*Pointer to the function to perform a optimization algorithm step.*
- static double(\* `optimize_estimate_climbing` )(unsigned int variable, unsigned int estimate)  
*Pointer to the function to estimate the climbing.*
- static double(\* `optimize_norm` )(unsigned int simulation)  
*Pointer to the error norm function.*

### 4.21.1 Detailed Description

Source file to define the optimization functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [optimize.c](#).

### 4.21.2 Macro Definition Documentation

#### 4.21.2.1 CP

```
#define CP "cp"
```

Macro to define the shell copy command.

Definition at line 79 of file [optimize.c](#).

#### 4.21.2.2 DEBUG\_OPTIMIZE

```
#define DEBUG_OPTIMIZE 0
```

Macro to debug optimize functions.

Definition at line 67 of file [optimize.c](#).

#### 4.21.2.3 RM

```
#define RM "rm"
```

Macro to define the shell remove command.

Definition at line 80 of file [optimize.c](#).

### 4.21.3 Function Documentation

#### 4.21.3.1 optimize\_best()

```
static void optimize_best (
    unsigned int simulation,
    double value ) [static]
```

Function to save the best simulations.

## Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 478 of file `optimize.c`.

```

00480 {
00481     unsigned int i, j;
00482     double e;
00483     #if DEBUG_OPTIMIZE
00484     fprintf (stderr, "optimize_best: start\n");
00485     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00486             optimize->nsaveds, optimize->nbest);
00487     #endif
00488     if (optimize->nsaveds < optimize->nbest
00489         || value < optimize->error_best[optimize->nsaveds - 1])
00490     {
00491         if (optimize->nsaveds < optimize->nbest)
00492             ++optimize->nsaveds;
00493         optimize->error_best[optimize->nsaveds - 1] = value;
00494         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00495         for (i = optimize->nsaveds; --i;)
00496         {
00497             if (optimize->error_best[i] < optimize->error_best[i - 1])
00498             {
00499                 j = optimize->simulation_best[i];
00500                 e = optimize->error_best[i];
00501                 optimize->simulation_best[i] = optimize->simulation_best[i - 1];
00502                 optimize->error_best[i] = optimize->error_best[i - 1];
00503                 optimize->simulation_best[i - 1] = j;
00504                 optimize->error_best[i - 1] = e;
00505             }
00506             else
00507                 break;
00508         }
00509     }
00510     #if DEBUG_OPTIMIZE
00511     fprintf (stderr, "optimize_best: end\n");
00512     #endif
00513 }

```

## 4.21.3.2 optimize\_best\_climbing()

```

static void optimize_best_climbing (
    unsigned int simulation,
    double value ) [static]

```

Function to save the best simulation in a hill climbing method.

## Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 847 of file `optimize.c`.

```

00849 {
00850     #if DEBUG_OPTIMIZE
00851     fprintf (stderr, "optimize_best_climbing: start\n");
00852     fprintf (stderr,
00853             "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00854             simulation, value, optimize->error_best[0]);
00855     #endif
00856     if (value < optimize->error_best[0])
00857     {
00858         optimize->error_best[0] = value;
00859         optimize->simulation_best[0] = simulation;
00860     }
00861     #if DEBUG_OPTIMIZE
00862     fprintf (stderr,
00863             "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00864             simulation, value);
00865     #endif
00866 }

```

```

00864 #endif
00865 }
00866 #if DEBUG_OPTIMIZE
00867     fprintf (stderr, "optimize_best_climbing: end\n");
00868 #endif
00869 }

```

### 4.21.3.3 optimize\_climbing()

```

static void optimize_climbing (
    unsigned int nsteps ) [inline], [static]

```

Function to optimize with a hill climbing method.

#### Parameters

<i>nsteps</i>	Number of steps.
---------------	------------------

Definition at line 1092 of file `optimize.c`.

```

01093 {
01094     unsigned int i, j, k, b, s, adjust;
01095     #if DEBUG_OPTIMIZE
01096         fprintf (stderr, "optimize_climbing: start\n");
01097     #endif
01098     for (i = 0; i < optimize->nvariables; ++i)
01099         optimize->climbing[i] = 0.;
01100     b = optimize->simulation_best[0] * optimize->nvariables;
01101     s = optimize->nsimulations;
01102     adjust = 1;
01103     for (i = 0; i < nsteps; ++i, s += optimize->nestimates, b = k)
01104     {
01105         #if DEBUG_OPTIMIZE
01106             fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01107                     i, optimize->simulation_best[0]);
01108         #endif
01109         optimize_step_climbing (s);
01110         k = optimize->simulation_best[0] * optimize->nvariables;
01111         #if DEBUG_OPTIMIZE
01112             fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01113                     i, optimize->simulation_best[0]);
01114         #endif
01115         if (k == b)
01116         {
01117             if (adjust)
01118                 for (j = 0; j < optimize->nvariables; ++j)
01119                     optimize->step[j] *= 0.5;
01120             for (j = 0; j < optimize->nvariables; ++j)
01121                 optimize->climbing[j] = 0.;
01122             adjust = 1;
01123         }
01124         else
01125         {
01126             for (j = 0; j < optimize->nvariables; ++j)
01127             {
01128                 #if DEBUG_OPTIMIZE
01129                     fprintf (stderr,
01130                             "optimize_climbing: best=%u old=%u\n",
01131                             j, optimize->value[k + j], j, optimize->value[b + j]);
01132                 #endif
01133                 optimize->climbing[j]
01134                     = (1. - optimize->relaxation) * optimize->climbing[j]
01135                     + optimize->relaxation
01136                     * (optimize->value[k + j] - optimize->value[b + j]);
01137                 #if DEBUG_OPTIMIZE
01138                     fprintf (stderr, "optimize_climbing: climbing=%u\n",
01139                             j, optimize->climbing[j]);
01140                 #endif
01141             }
01142             adjust = 0;
01143         }
01144     }
01145     #if DEBUG_OPTIMIZE
01146         fprintf (stderr, "optimize_climbing: end\n");
01147     #endif
01148 }

```

Here is the call graph for this function:



#### 4.21.3.4 optimize\_climbing\_best()

```
static void optimize_climbing_best ( ) [inline], [static]
```

Function to select the best simulation to start the hill climbing method.

Definition at line 1075 of file [optimize.c](#).

```

01076 {
01077     #if DEBUG_OPTIMIZE
01078         fprintf (stderr, "optimize_climbing_best: start\n");
01079     #endif
01080     optimize->simulation_best[0] = 0;
01081     memcpy (optimize->value, optimize->value_old,
01082            optimize->nvariables * sizeof (double));
01083     #if DEBUG_OPTIMIZE
01084         fprintf (stderr, "optimize_climbing_best: end\n");
01085     #endif
01086 }
```

#### 4.21.3.5 optimize\_climbing\_sequential()

```
static void optimize_climbing_sequential (
    unsigned int simulation ) [inline], [static]
```

Function to estimate the hill climbing sequentially.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 875 of file [optimize.c](#).

```

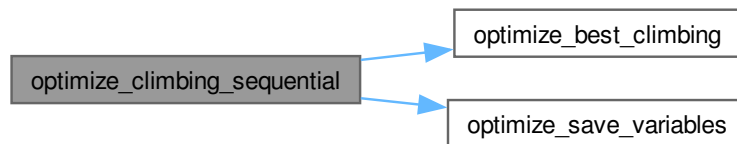
00876 {
00877     double e;
00878     unsigned int i, j;
00879     #if DEBUG_OPTIMIZE
00880         fprintf (stderr, "optimize_climbing_sequential: start\n");
00881         fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00882            "nend_climbing=%u\n",
00883            optimize->nstart_climbing, optimize->nend_climbing);
00884     #endif
00885     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00886     {
00887         j = simulation + i;
00888         e = optimize_norm (j);
00889         optimize_best_climbing (j, e);
00890         optimize_save_variables (j, e);
00891         if (e < optimize->threshold)
00892         {
00893             optimize->stop = 1;
00894             break;
00895         }
00896     }
00897     #if DEBUG_OPTIMIZE
00898         fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
00899     #endif
00900 }
```

```

00898 #endif
00899 }
00900 #if DEBUG_OPTIMIZE
00901     fprintf (stderr, "optimize_climbing_sequential: end\n");
00902 #endif
00903 }

```

Here is the call graph for this function:



#### 4.21.3.6 optimize\_climbing\_thread()

```

static void * optimize_climbing_thread (
    ParallelData * data ) [static]

```

Function to estimate the hill climbing on a thread.

##### Returns

NULL

##### Parameters

<i>data</i>	Function data.
-------------	----------------

Definition at line 911 of file [optimize.c](#).

```

00912 {
00913     unsigned int i, thread;
00914     double e;
00915     #if DEBUG_OPTIMIZE
00916         fprintf (stderr, "optimize_climbing_thread: start\n");
00917     #endif
00918     thread = data->thread;
00919     #if DEBUG_OPTIMIZE
00920         fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00921             thread,
00922             optimize->thread_climbing[thread],
00923             optimize->thread_climbing[thread + 1]);
00924     #endif
00925     for (i = optimize->thread_climbing[thread];
00926         i < optimize->thread_climbing[thread + 1]; ++i)
00927     {
00928         e = optimize_norm (i);
00929         g_mutex_lock (mutex);
00930         optimize_best_climbing (i, e);
00931         optimize_save_variables (i, e);
00932         if (e < optimize->threshold)
00933             optimize->stop = 1;
00934         g_mutex_unlock (mutex);
00935         if (optimize->stop)
00936             break;
00937     #if DEBUG_OPTIMIZE

```

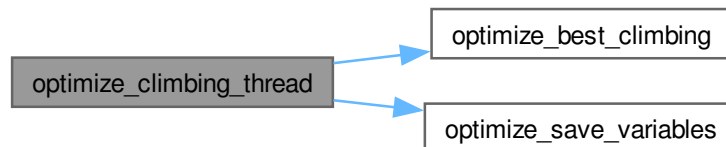


```

00938     fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00939 #endif
00940 }
00941 #if DEBUG_OPTIMIZE
00942 fprintf (stderr, "optimize_climbing_thread: end\n");
00943 #endif
00944 g_thread_exit (NULL);
00945 return NULL;
00946 }

```

Here is the call graph for this function:



#### 4.21.3.7 optimize\_estimate\_climbing\_coordinates()

```

static double optimize_estimate_climbing_coordinates (
    unsigned int variable,
    unsigned int estimate ) [static]

```

Function to estimate a component of the hill climbing vector.

##### Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 976 of file [optimize.c](#).

```

00980 {
00981     double x;
00982     #if DEBUG_OPTIMIZE
00983     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00984     #endif
00985     x = optimize->climbing[variable];
00986     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00987     {
00988         if (estimate & 1)
00989             x += optimize->step[variable];
00990         else
00991             x -= optimize->step[variable];
00992     }
00993     #if DEBUG_OPTIMIZE
00994     fprintf (stderr,
00995         "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
00996         variable, x);
00997     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00998     #endif
00999     return x;
01000 }

```

#### 4.21.3.8 optimize\_estimate\_climbing\_random()

```

static double optimize_estimate_climbing_random (

```

```
unsigned int variable,
unsigned int estimate ) [static]
```

Function to estimate a component of the hill climbing vector.

#### Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 952 of file [optimize.c](#).

```
00957 {
00958     double x;
00959     #if DEBUG_OPTIMIZE
00960     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00961     #endif
00962     x = optimize->climbing[variable]
00963         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00964     #if DEBUG_OPTIMIZE
00965     fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
00966             variable, x);
00967     fprintf (stderr, "optimize_estimate_climbing_random: end\n");
00968     #endif
00969     return x;
00970 }
```

#### 4.21.3.9 optimize\_free()

```
void optimize_free ( )
```

Function to free the memory used by the [Optimize](#) struct.

Definition at line 1487 of file [optimize.c](#).

```
01488 {
01489     unsigned int i, j;
01490     #if DEBUG_OPTIMIZE
01491     fprintf (stderr, "optimize_free: start\n");
01492     #endif
01493     for (j = 0; j < optimize->ninputs; ++j)
01494     {
01495         for (i = 0; i < optimize->nexperiments; ++i)
01496             g_mapped_file_unref (optimize->file[j][i]);
01497         g_free (optimize->file[j]);
01498     }
01499     g_free (optimize->error_old);
01500     g_free (optimize->value_old);
01501     g_free (optimize->value);
01502     g_free (optimize->genetic_variable);
01503     #if DEBUG_OPTIMIZE
01504     fprintf (stderr, "optimize_free: end\n");
01505     #endif
01506 }
```

#### 4.21.3.10 optimize\_genetic()

```
static void optimize_genetic ( ) [static]
```

Function to optimize with the genetic algorithm.

Definition at line 1189 of file [optimize.c](#).

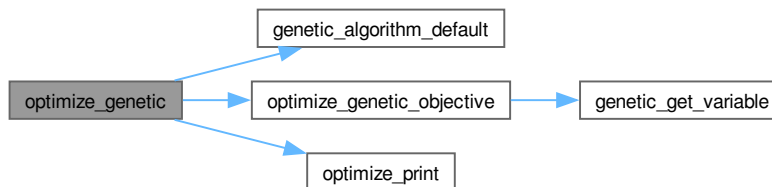
```
01190 {
01191     double *best_variable = NULL;
01192     char *best_genome = NULL;
01193     double best_objective = 0.;
01194     #if DEBUG_OPTIMIZE
01195     fprintf (stderr, "optimize_genetic: start\n");
01196     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
```

```

01197         nthreads);
01198     fprintf (stderr,
01199             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01200             optimize->nvariables, optimize->nsimulations, optimize->niterations);
01201     fprintf (stderr,
01202             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01203             optimize->mutation_ratio, optimize->reproduction_ratio,
01204             optimize->adaptation_ratio);
01205 #endif
01206     genetic_algorithm_default (optimize->nvariables,
01207                               optimize->genetic_variable,
01208                               optimize->nsimulations,
01209                               optimize->niterations,
01210                               optimize->mutation_ratio,
01211                               optimize->reproduction_ratio,
01212                               optimize->adaptation_ratio,
01213                               optimize->seed,
01214                               optimize->threshold,
01215                               &optimize_genetic_objective,
01216                               &best_genome, &best_variable, &best_objective);
01217 #if DEBUG_OPTIMIZE
01218     fprintf (stderr, "optimize_genetic: the best\n");
01219 #endif
01220     optimize->error_old = (double *) g_malloc (sizeof (double));
01221     optimize->value_old
01222         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01223     optimize->error_old[0] = best_objective;
01224     memcpy (optimize->value_old, best_variable,
01225            optimize->nvariables * sizeof (double));
01226     g_free (best_genome);
01227     g_free (best_variable);
01228     optimize_print ();
01229 #if DEBUG_OPTIMIZE
01230     fprintf (stderr, "optimize_genetic: end\n");
01231 #endif
01232 }

```

Here is the call graph for this function:



#### 4.21.3.11 optimize\_genetic\_objective()

```

static double optimize_genetic_objective (
    Entity * entity ) [static]

```

Function to calculate the objective function of an entity.

##### Returns

objective function value.

##### Parameters

<i>entity</i>	entity data.
---------------	--------------

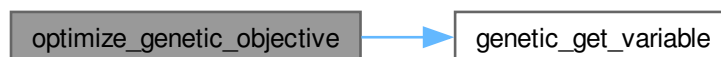
Definition at line 1156 of file [optimize.c](#).

```

01157 {
01158     unsigned int j;
01159     double objective;
01160     char buffer[64];
01161     #if DEBUG_OPTIMIZE
01162     fprintf (stderr, "optimize_genetic_objective: start\n");
01163     #endif
01164     for (j = 0; j < optimize->nvariables; ++j)
01165     {
01166         optimize->value[entity->id * optimize->nvariables + j]
01167         = genetic_get_variable (entity, optimize->genetic_variable + j);
01168     }
01169     objective = optimize_norm (entity->id);
01170     g_mutex_lock (mutex);
01171     for (j = 0; j < optimize->nvariables; ++j)
01172     {
01173         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01174         fprintf (optimize->file_variables, buffer,
01175                 genetic_get_variable (entity, optimize->genetic_variable + j));
01176     }
01177     fprintf (optimize->file_variables, "%.14le\n", objective);
01178     g_mutex_unlock (mutex);
01179     #if DEBUG_OPTIMIZE
01180     fprintf (stderr, "optimize_genetic_objective: end\n");
01181     #endif
01182     return objective;
01183 }

```

Here is the call graph for this function:



#### 4.21.3.12 optimize\_input()

```

static void optimize_input (
    unsigned int simulation,
    char * input,
    GMappedFile * stencil ) [inline], [static]

```

Function to write the simulation input file.

##### Parameters

<i>simulation</i>	Simulation number.
<i>input</i>	<a href="#">Input</a> file name.
<i>stencil</i>	Template of the input file name.

Definition at line 99 of file [optimize.c](#).

```

00102 {
00103     char buffer[256], value[32];
00104     GRegex *regex;
00105     FILE *file;
00106     char *buffer2, *buffer3 = NULL, *content;
00107     gsize length;
00108     unsigned int i;
00109 }

```

```

00110 #if DEBUG_OPTIMIZE
00111     fprintf (stderr, "optimize_input: start\n");
00112 #endif
00113
00114     // Checking the file
00115     if (!stencil)
00116         goto optimize_input_end;
00117
00118     // Opening stencil
00119     content = g_mapped_file_get_contents (stencil);
00120     length = g_mapped_file_get_length (stencil);
00121 #if DEBUG_OPTIMIZE
00122     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00123 #endif
00124     file = g_fopen (input, "w");
00125
00126     // Parsing stencil
00127     for (i = 0; i < optimize->nvariables; ++i)
00128     {
00129 #if DEBUG_OPTIMIZE
00130         fprintf (stderr, "optimize_input: variable=%u\n", i);
00131 #endif
00132         snprintf (buffer, 32, "@variable%u@", i + 1);
00133         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00134                               NULL);
00135         if (i == 0)
00136         {
00137             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00138                                                optimize->label[i],
00139                                                (GRegexMatchFlags) 0, NULL);
00140 #if DEBUG_OPTIMIZE
00141             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00142 #endif
00143         }
00144         else
00145         {
00146             length = strlen (buffer3);
00147             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00148                                                optimize->label[i],
00149                                                (GRegexMatchFlags) 0, NULL);
00150             g_free (buffer3);
00151         }
00152         g_regex_unref (regex);
00153         length = strlen (buffer2);
00154         snprintf (buffer, 32, "@value%u@", i + 1);
00155         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00156                               NULL);
00157         snprintf (value, 32, format[optimize->precision[i]],
00158                  optimize->value[simulation * optimize->nvariables + i]);
00159 #if DEBUG_OPTIMIZE
00160         fprintf (stderr, "optimize_input: value=%s\n", value);
00161 #endif
00162         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                           (GRegexMatchFlags) 0, NULL);
00164         g_free (buffer2);
00165         g_regex_unref (regex);
00166     }
00167
00168     // Saving input file
00169     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170     g_free (buffer3);
00171     fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177     return;
00178 }
00179

```

#### 4.21.3.13 optimize\_iterate()

```
static void optimize_iterate ( ) [inline], [static]
```

Function to iterate the algorithm.

Definition at line 1420 of file [optimize.c](#).

```

01421 {
01422     unsigned int i;
01423 #if DEBUG_OPTIMIZE

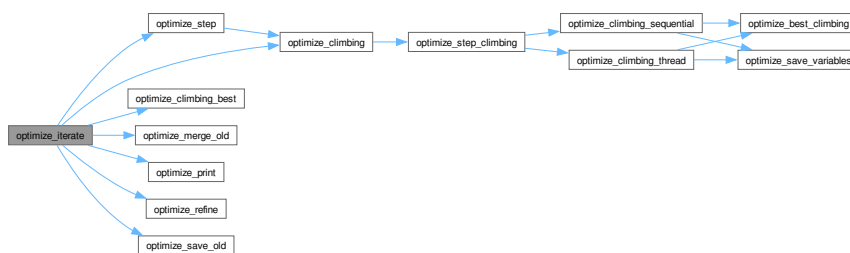
```

```

01424 fprintf (stderr, "optimize_iterate: start\n");
01425 #endif
01426 optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01427 optimize->value_old =
01428     (double *) g_malloc (optimize->nbest * optimize->nvariables *
01429         sizeof (double));
01430 optimize_step ();
01431 optimize_save_old ();
01432 optimize_refine ();
01433 optimize_print ();
01434 for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01435 {
01436     optimize_step ();
01437     optimize_merge_old ();
01438     optimize_refine ();
01439     optimize_print ();
01440 }
01441 if (optimize->nfinal_steps && !optimize->stop)
01442 {
01443     optimize_climbing_best ();
01444     optimize_climbing (optimize->nfinal_steps);
01445     optimize_merge_old ();
01446     optimize_print ();
01447 }
01448 #if DEBUG_OPTIMIZE
01449 fprintf (stderr, "optimize_iterate: end\n");
01450 #endif
01451 }

```

Here is the call graph for this function:



#### 4.21.3.14 optimize\_merge()

```

static void optimize_merge (
    unsigned int nsaveds,
    unsigned int * simulation_best,
    double * error_best ) [inline], [static]

```

Function to merge the 2 optimization results.

##### Parameters

<i>nsaveds</i>	Number of saved results.
<i>simulation_best</i>	Array of best simulation numbers.
<i>error_best</i>	Array of best objective function values.

Definition at line 591 of file [optimize.c](#).

```

00596 {
00597     unsigned int i, j, k, s[optimize->nbest];
00598     double e[optimize->nbest];
00599     #if DEBUG_OPTIMIZE
00600     fprintf (stderr, "optimize_merge: start\n");

```

```

00601 #endif
00602     i = j = k = 0;
00603     do
00604     {
00605         if (i == optimize->nsaveds)
00606         {
00607             s[k] = simulation_best[j];
00608             e[k] = error_best[j];
00609             ++j;
00610             ++k;
00611             if (j == nsaveds)
00612                 break;
00613         }
00614         else if (j == nsaveds)
00615         {
00616             s[k] = optimize->simulation_best[i];
00617             e[k] = optimize->error_best[i];
00618             ++i;
00619             ++k;
00620             if (i == optimize->nsaveds)
00621                 break;
00622         }
00623         else if (optimize->error_best[i] > error_best[j])
00624         {
00625             s[k] = simulation_best[j];
00626             e[k] = error_best[j];
00627             ++j;
00628             ++k;
00629         }
00630         else
00631         {
00632             s[k] = optimize->simulation_best[i];
00633             e[k] = optimize->error_best[i];
00634             ++i;
00635             ++k;
00636         }
00637     }
00638     while (k < optimize->nbest);
00639     optimize->nsaveds = k;
00640     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00641     memcpy (optimize->error_best, e, k * sizeof (double));
00642     #if DEBUG_OPTIMIZE
00643     fprintf (stderr, "optimize_merge: end\n");
00644 #endif
00645 }

```

#### 4.21.3.15 optimize\_merge\_old()

```
static void optimize_merge_old ( ) [inline], [static]
```

Function to merge the best results with the previous step best results on iterative methods.

Definition at line 1270 of file [optimize.c](#).

```

01271 {
01272     unsigned int i, j, k;
01273     double v[optimize->nbest * optimize->nvariables], e[optimize->nbest],
01274           *enew, *eold;
01275     #if DEBUG_OPTIMIZE
01276     fprintf (stderr, "optimize_merge_old: start\n");
01277 #endif
01278     anew = optimize->error_best;
01279     eold = optimize->error_old;
01280     i = j = k = 0;
01281     do
01282     {
01283         if (*enew < *eold)
01284         {
01285             memcpy (v + k * optimize->nvariables,
01286                     optimize->value
01287                     + optimize->simulation_best[i] * optimize->nvariables,
01288                     optimize->nvariables * sizeof (double));
01289             e[k] = *enew;
01290             ++k;
01291             ++enew;
01292             ++i;
01293         }
01294         else
01295         {
01296             memcpy (v + k * optimize->nvariables,
01297                     optimize->value_old + j * optimize->nvariables,
01298                     optimize->nvariables * sizeof (double));

```

```

01299         e[k] = *eold;
01300         ++k;
01301         ++eold;
01302         ++j;
01303     }
01304 }
01305 while (k < optimize->nbest);
01306 memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01307 memcpy (optimize->error_old, e, k * sizeof (double));
01308 #if DEBUG_OPTIMIZE
01309 fprintf (stderr, "optimize_merge_old: end\n");
01310 #endif
01311 }

```

#### 4.21.3.16 optimize\_MonteCarlo()

```
static void optimize_MonteCarlo ( ) [static]
```

Function to optimize with the Monte-Carlo algorithm.

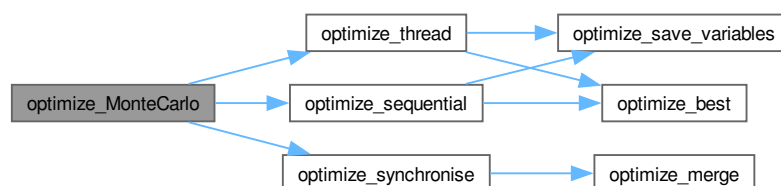
Definition at line 752 of file [optimize.c](#).

```

00753 {
00754     ParallelData data[nthreads];
00755     GThread *thread[nthreads];
00756     double range[optimize->nvariables];
00757     unsigned int i, j;
00758 #if DEBUG_OPTIMIZE
00759 fprintf (stderr, "optimize_MonteCarlo: start\n");
00760 #endif
00761 for (j = 0; j < optimize->nvariables; ++j)
00762     range[j] = optimize->rangemax[j] - optimize->rangemin[j];
00763 for (i = 0; i < optimize->nsimulations; ++i)
00764     for (j = 0; j < optimize->nvariables; ++j)
00765         optimize->value[i * optimize->nvariables + j]
00766             = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng) * range[j];
00767 optimize->nsaveds = 0;
00768 if (nthreads <= 1)
00769     optimize_sequential ();
00770 else
00771     {
00772         for (i = 0; i < nthreads; ++i)
00773         {
00774             data[i].thread = i;
00775             thread[i]
00776                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00777         }
00778         for (i = 0; i < nthreads; ++i)
00779             g_thread_join (thread[i]);
00780     }
00781 #if HAVE_MPI
00782 // Communicating tasks results
00783 optimize_synchronise ();
00784 #endif
00785 #if DEBUG_OPTIMIZE
00786 fprintf (stderr, "optimize_MonteCarlo: end\n");
00787 #endif
00788 }

```

Here is the call graph for this function:





## 4.21.3.17 optimize\_norm\_euclidian()

```
static double optimize_norm_euclidian (
    unsigned int simulation ) [static]
```

Function to calculate the Euclidian error norm.

## Returns

Euclidian error norm.

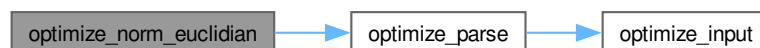
## Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 326 of file [optimize.c](#).

```
00327 {
00328     double e, ei;
00329     unsigned int i;
00330     #if DEBUG_OPTIMIZE
00331     fprintf (stderr, "optimize_norm_euclidian: start\n");
00332     #endif
00333     e = 0.;
00334     for (i = 0; i < optimize->nexperiments; ++i)
00335     {
00336         ei = optimize_parse (simulation, i);
00337         e += ei * ei;
00338     }
00339     e = sqrt (e);
00340     #if DEBUG_OPTIMIZE
00341     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00342     fprintf (stderr, "optimize_norm_euclidian: end\n");
00343     #endif
00344     return e;
00345 }
```

Here is the call graph for this function:



## 4.21.3.18 optimize\_norm\_maximum()

```
static double optimize_norm_maximum (
    unsigned int simulation ) [static]
```

Function to calculate the maximum error norm.

## Returns

Maximum error norm.

## Parameters

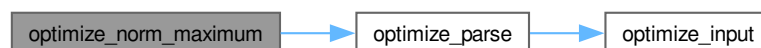
<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 353 of file [optimize.c](#).

```

00354 {
00355     double e, ei;
00356     unsigned int i;
00357     #if DEBUG_OPTIMIZE
00358     fprintf (stderr, "optimize_norm_maximum: start\n");
00359     #endif
00360     e = 0.;
00361     for (i = 0; i < optimize->nexperiments; ++i)
00362     {
00363         ei = fabs (optimize_parse (simulation, i));
00364         e = fmax (e, ei);
00365     }
00366     #if DEBUG_OPTIMIZE
00367     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00368     fprintf (stderr, "optimize_norm_maximum: end\n");
00369     #endif
00370     return e;
00371 }
```

Here is the call graph for this function:



#### 4.21.3.19 optimize\_norm\_p()

```

static double optimize_norm_p (
    unsigned int simulation ) [static]
```

Function to calculate the P error norm.

## Returns

P error norm.

## Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 379 of file [optimize.c](#).

```

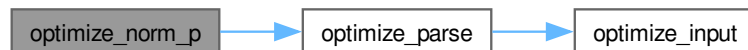
00380 {
00381     double e, ei;
00382     unsigned int i;
00383     #if DEBUG_OPTIMIZE
00384     fprintf (stderr, "optimize_norm_p: start\n");
00385     #endif
00386     e = 0.;
00387     for (i = 0; i < optimize->nexperiments; ++i)
00388     {
00389         ei = fabs (optimize_parse (simulation, i));
00390         e += pow (ei, optimize->p);
00391     }
```

```

00391     }
00392     e = pow (e, 1. / optimize->p);
00393     #if DEBUG_OPTIMIZE
00394     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00395     fprintf (stderr, "optimize_norm_p: end\n");
00396     #endif
00397     return e;
00398 }

```

Here is the call graph for this function:



#### 4.21.3.20 optimize\_norm\_taxicab()

```

static double optimize_norm_taxicab (
    unsigned int simulation ) [static]

```

Function to calculate the taxicab error norm.

##### Returns

Taxicab error norm.

##### Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

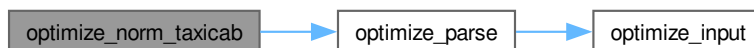
Definition at line 406 of file [optimize.c](#).

```

00407 {
00408     double e;
00409     unsigned int i;
00410     #if DEBUG_OPTIMIZE
00411     fprintf (stderr, "optimize_norm_taxicab: start\n");
00412     #endif
00413     e = 0.;
00414     for (i = 0; i < optimize->nexperiments; ++i)
00415         e += fabs (optimize_parse (simulation, i));
00416     #if DEBUG_OPTIMIZE
00417     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00418     fprintf (stderr, "optimize_norm_taxicab: end\n");
00419     #endif
00420     return e;
00421 }

```

Here is the call graph for this function:



#### 4.21.3.21 optimize\_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1512 of file [optimize.c](#).

```
01513 {
01514     GTimeZone *tz;
01515     GDateTime *t0, *t;
01516     unsigned int i, j, nsteps;
01517
01518     #if DEBUG_OPTIMIZE
01519         char *buffer;
01520         fprintf (stderr, "optimize_open: start\n");
01521     #endif
01522
01523     // Getting initial time
01524     #if DEBUG_OPTIMIZE
01525         fprintf (stderr, "optimize_open: getting initial time\n");
01526     #endif
01527     tz = g_time_zone_new_utc ();
01528     t0 = g_date_time_new_now (tz);
01529
01530     // Obtaining and initing the pseudo-random numbers generator seed
01531     #if DEBUG_OPTIMIZE
01532         fprintf (stderr, "optimize_open: getting initial seed\n");
01533     #endif
01534     if (optimize->seed == DEFAULT_RANDOM_SEED)
01535         optimize->seed = input->seed;
01536     gsl_rng_set (optimize->rng, optimize->seed);
01537
01538     // Obtaining template flags
01539     #if DEBUG_OPTIMIZE
01540         fprintf (stderr, "optimize_open: getting template flags\n");
01541     #endif
01542     optimize->template_flags = input->template_flags;
01543
01544     // Replacing the working directory
01545     #if DEBUG_OPTIMIZE
01546         fprintf (stderr, "optimize_open: replacing the working directory\n");
01547     #endif
01548     g_chdir (input->directory);
01549
01550     // Getting results file names
01551     optimize->result = input->result;
01552     optimize->variables = input->variables;
01553
01554     // Obtaining the simulator file
01555     optimize->simulator = input->simulator;
01556
01557     // Obtaining the evaluator file
01558     optimize->evaluator = input->evaluator;
01559
01560     // Obtaining the cleaner file
01561     optimize->cleaner = input->cleaner;
01562
01563     // Reading the algorithm
01564     optimize->algorithm = input->algorithm;
01565     switch (optimize->algorithm)
01566     {
01567         case ALGORITHM_MONTE_CARLO:
01568             optimize_algorithm = optimize_MonteCarlo;
01569             break;
01570         case ALGORITHM_SWEEP:
01571             optimize_algorithm = optimize_sweep;
01572             break;
01573         case ALGORITHM_ORTHOGONAL:
01574             optimize_algorithm = optimize_orthogonal;
01575             break;
01576         default:
01577             optimize_algorithm = optimize_genetic;
01578             optimize->mutation_ratio = input->mutation_ratio;
01579             optimize->reproduction_ratio = input->reproduction_ratio;
01580             optimize->adaptation_ratio = input->adaptation_ratio;
01581     }
01582     optimize->nvariables = input->nvariables;
01583     optimize->nsimulations = input->nsimulations;
01584     optimize->niterations = input->niterations;
01585     optimize->nbest = input->nbest;
01586     optimize->tolerance = input->tolerance;
01587     optimize->nsteps = input->nsteps;
01588     optimize->nfinal_steps = input->nfinal_steps;
```

```

01589     nsteps = JBM_MAX (optimize->nsteps, optimize->nfinal_steps);
01590     optimize->nestimates = 0;
01591     optimize->threshold = input->threshold;
01592     optimize->stop = 0;
01593     if (nsteps)
01594     {
01595         optimize->relaxation = input->relaxation;
01596         switch (input->climbing)
01597         {
01598             case CLIMBING_METHOD_COORDINATES:
01599                 optimize->nestimates = 2 * optimize->nvariables;
01600                 optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
01601                 break;
01602             default:
01603                 optimize->nestimates = input->nestimates;
01604                 optimize_estimate_climbing = optimize_estimate_climbing_random;
01605         }
01606     }
01607
01608     #if DEBUG_OPTIMIZE
01609     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01610     #endif
01611     optimize->simulation_best
01612     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01613     optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01614
01615     // Reading the experimental data
01616     #if DEBUG_OPTIMIZE
01617     buffer = g_get_current_dir ();
01618     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01619     g_free (buffer);
01620     #endif
01621     optimize->nexperiments = input->nexperiments;
01622     optimize->ninputs = input->experiment->ninputs;
01623     optimize->experiment
01624     = (char **) alloca (input->nexperiments * sizeof (char *));
01625     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01626     for (i = 0; i < input->experiment->ninputs; ++i)
01627         optimize->file[i] = (GMappedFile **)
01628         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01629     for (i = 0; i < input->nexperiments; ++i)
01630     {
01631         #if DEBUG_OPTIMIZE
01632         fprintf (stderr, "optimize_open: i=%u\n", i);
01633         #endif
01634         optimize->experiment[i] = input->experiment[i].name;
01635         optimize->weight[i] = input->experiment[i].weight;
01636         #if DEBUG_OPTIMIZE
01637         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01638                 optimize->experiment[i], optimize->weight[i]);
01639         #endif
01640         for (j = 0; j < input->experiment->ninputs; ++j)
01641         {
01642             #if DEBUG_OPTIMIZE
01643             fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01644             #endif
01645             optimize->file[j][i]
01646             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01647         }
01648     }
01649
01650     // Reading the variables data
01651     #if DEBUG_OPTIMIZE
01652     fprintf (stderr, "optimize_open: reading variables\n");
01653     #endif
01654     optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01655     j = input->nvariables * sizeof (double);
01656     optimize->rangemin = (double *) alloca (j);
01657     optimize->rangeminabs = (double *) alloca (j);
01658     optimize->rangemax = (double *) alloca (j);
01659     optimize->rangemaxabs = (double *) alloca (j);
01660     optimize->step = (double *) alloca (j);
01661     j = input->nvariables * sizeof (unsigned int);
01662     optimize->precision = (unsigned int *) alloca (j);
01663     optimize->nsweeps = (unsigned int *) alloca (j);
01664     optimize->nbits = (unsigned int *) alloca (j);
01665     for (i = 0; i < input->nvariables; ++i)
01666     {
01667         optimize->label[i] = input->variable[i].name;
01668         optimize->rangemin[i] = input->variable[i].rangemin;
01669         optimize->rangeminabs[i] = input->variable[i].rangeminabs;
01670         optimize->rangemax[i] = input->variable[i].rangemax;
01671         optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
01672         optimize->precision[i] = input->variable[i].precision;
01673         optimize->step[i] = input->variable[i].step;
01674         optimize->nsweeps[i] = input->variable[i].nsweeps;
01675         optimize->nbits[i] = input->variable[i].nbits;

```

```

01676     }
01677     if (input->algorithm == ALGORITHM_SWEEP
01678         || input->algorithm == ALGORITHM_ORTHOGONAL)
01679     {
01680         optimize->nsimulations = 1;
01681         for (i = 0; i < input->nvariables; ++i)
01682         {
01683             optimize->nsimulations *= optimize->nsweeps[i];
01684 #if DEBUG_OPTIMIZE
01685             fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01686                     optimize->nsweeps[i], optimize->nsimulations);
01687 #endif
01688         }
01689     }
01690     if (nsteps)
01691         optimize->climbing
01692         = (double *) alloca (optimize->nvariables * sizeof (double));
01693
01694     // Setting error norm
01695     switch (input->norm)
01696     {
01697     case ERROR_NORM_EUCLIDIAN:
01698         optimize_norm = optimize_norm_euclidian;
01699         break;
01700     case ERROR_NORM_MAXIMUM:
01701         optimize_norm = optimize_norm_maximum;
01702         break;
01703     case ERROR_NORM_P:
01704         optimize_norm = optimize_norm_p;
01705         optimize->p = input->p;
01706         break;
01707     default:
01708         optimize_norm = optimize_norm_taxicab;
01709     }
01710
01711     // Allocating values
01712 #if DEBUG_OPTIMIZE
01713     fprintf (stderr, "optimize_open: allocating variables\n");
01714     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01715             optimize->nvariables, optimize->algorithm);
01716 #endif
01717     optimize->genetic_variable = NULL;
01718     if (optimize->algorithm == ALGORITHM_GENETIC)
01719     {
01720         optimize->genetic_variable = (GeneticVariable *)
01721             g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01722         for (i = 0; i < optimize->nvariables; ++i)
01723         {
01724 #if DEBUG_OPTIMIZE
01725             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01726                     i, optimize->rangemin[i], optimize->rangemax[i],
01727                     optimize->nbits[i]);
01728 #endif
01729             optimize->genetic_variable[i].minimum = optimize->rangemin[i];
01730             optimize->genetic_variable[i].maximum = optimize->rangemax[i];
01731             optimize->genetic_variable[i].nbits = optimize->nbits[i];
01732         }
01733     }
01734 #if DEBUG_OPTIMIZE
01735     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01736             optimize->nvariables, optimize->nsimulations);
01737 #endif
01738     optimize->value = (double *)
01739         g_malloc ((optimize->nsimulations + optimize->nestimates * nsteps)
01740                 * optimize->nvariables * sizeof (double));
01741
01742     // Calculating simulations to perform for each task
01743 #if HAVE_MPI
01744 #if DEBUG_OPTIMIZE
01745     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01746             optimize->mpi_rank, ntasks);
01747 #endif
01748     optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
01749     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
01750     if (nsteps)
01751     {
01752         optimize->nstart_climbing
01753         = optimize->mpi_rank * optimize->nestimates / ntasks;
01754         optimize->nend_climbing
01755         = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
01756     }
01757 #else
01758     optimize->nstart = 0;
01759     optimize->nend = optimize->nsimulations;
01760     if (nsteps)
01761     {
01762         optimize->nstart_climbing = 0;

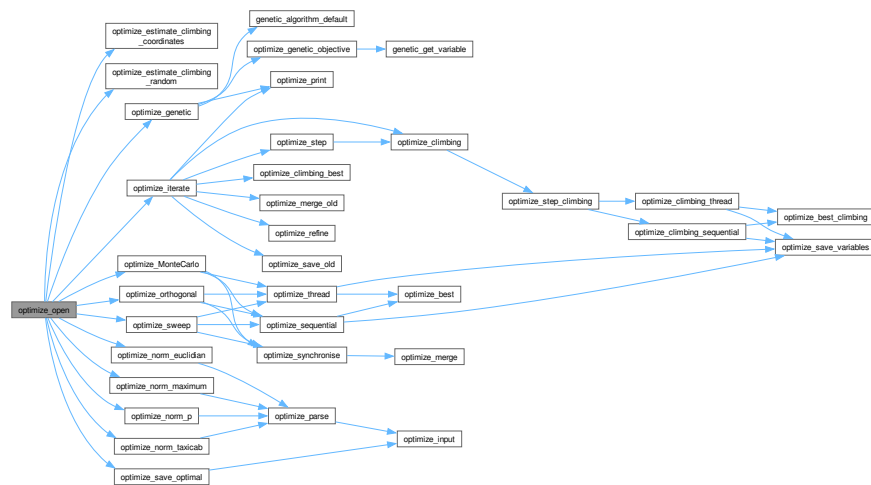
```

```

01763     optimize->nend_climbing = optimize->nestimates;
01764 }
01765 #endif
01766 #if DEBUG_OPTIMIZE
01767 fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01768         optimize->nend);
01769 #endif
01770
01771 // Calculating simulations to perform for each thread
01772 optimize->thread
01773 = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01774 for (i = 0; i <= nthreads; ++i)
01775 {
01776     optimize->thread[i] = optimize->nstart
01777         + i * (optimize->nend - optimize->nstart) / nthreads;
01778 #if DEBUG_OPTIMIZE
01779 fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01780         optimize->thread[i]);
01781 #endif
01782 }
01783 if (nsteps)
01784     optimize->thread_climbing = (unsigned int *)
01785         alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01786
01787 // Opening result files
01788 optimize->file_result = g_fopen (optimize->result, "w");
01789 optimize->file_variables = g_fopen (optimize->variables, "w");
01790
01791 // Performing the algorithm
01792 switch (optimize->algorithm)
01793 {
01794     // Genetic algorithm
01795     case ALGORITHM_GENETIC:
01796         optimize_genetic ();
01797         break;
01798     // Iterative algorithm
01799     default:
01800         optimize_iterate ();
01801 }
01802
01803 // Getting calculation time
01804 t = g_date_time_new_now (tz);
01805 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01806 g_date_time_unref (t);
01807 g_date_time_unref (t0);
01808 g_time_zone_unref (tz);
01809 printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01810 fprintf (optimize->file_result, "%s = %.6lg s\n",
01811         _("Calculation time"), optimize->calculation_time);
01812
01813 // Closing result files
01814 optimize_save_optimal ();
01815 fclose (optimize->file_variables);
01816 fclose (optimize->file_result);
01817
01818 #if DEBUG_OPTIMIZE
01819 fprintf (stderr, "optimize_open: end\n");
01820 #endif
01821 }

```

Here is the call graph for this function:



#### 4.21.3.22 optimize\_orthogonal()

```
static void optimize_orthogonal ( ) [static]
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 794 of file [optimize.c](#).

```

00795 {
00796     ParallelData data[nthreads];
00797     GThread *thread[nthreads];
00798     double range[optimize->nvariables];
00799     double e;
00800     unsigned int i, j, k, l;
00801     #if DEBUG_OPTIMIZE
00802     fprintf (stderr, "optimize_orthogonal: start\n");
00803     #endif
00804     for (j = 0; j < optimize->nvariables; ++j)
00805         range[j] = (optimize->rangemax[j] - optimize->rangemin[j])
00806             / optimize->nsweeps[j];
00807     for (i = 0; i < optimize->nsimulations; ++i)
00808     {
00809         k = i;
00810         for (j = 0; j < optimize->nvariables; ++j)
00811         {
00812             l = k % optimize->nsweeps[j];
00813             k /= optimize->nsweeps[j];
00814             e = optimize->rangemin[j];
00815             if (optimize->nsweeps[j] > 1)
00816                 e += (l + gsl_rng_uniform (optimize->rng)) * range[j];
00817             optimize->value[i * optimize->nvariables + j] = e;
00818         }
00819     }
00820     optimize->nsaveds = 0;
00821     if (nthreads <= 1)
00822         optimize_sequential ();
00823     else
00824     {
00825         for (i = 0; i < nthreads; ++i)
00826         {
00827             data[i].thread = i;
00828             thread[i]
00829                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00830         }
00831         for (i = 0; i < nthreads; ++i)
00832             g_thread_join (thread[i]);
00833     }
00834     #if HAVE_MPI
00835     // Communicating tasks results
00836     optimize_synchronise ();

```

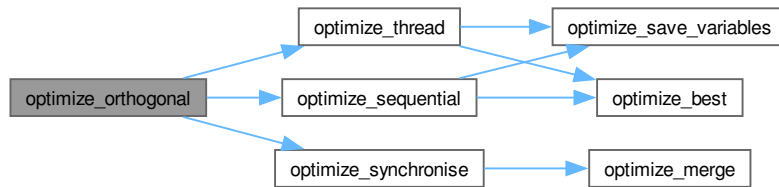


```

00837 #endif
00838 #if DEBUG_OPTIMIZE
00839     fprintf (stderr, "optimize_orthogonal: end\n");
00840 #endif
00841 }

```

Here is the call graph for this function:



#### 4.21.3.23 optimize\_parse()

```

static double optimize_parse (
    unsigned int simulation,
    unsigned int experiment ) [static]

```

Function to parse input files, simulating and calculating the objective function.

##### Returns

Objective function value.

##### Parameters

<i>simulation</i>	Simulation number.
<i>experiment</i>	Experiment number.

Definition at line 188 of file [optimize.c](#).

```

00190 {
00191     char buffer[512], cinput[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00192         *buffer3, *buffer4;
00193     FILE *file_result;
00194     double e;
00195     unsigned int i;
00196     unsigned int flags = 1;
00197
00198     #if DEBUG_OPTIMIZE
00199         fprintf (stderr, "optimize_parse: start\n");
00200         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00201                 simulation, experiment);
00202     #endif
00203
00204     // Opening input files
00205     for (i = 0; i < optimize->ninputs; ++i)
00206     {
00207         snprintf (&cinput[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00208     #if DEBUG_OPTIMIZE
00209         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &cinput[i][0]);
00210     #endif
00211     // Checking simple copy
00212     if (optimize->template_flags & flags)
00213         optimize_input (simulation, &cinput[i][0],

```

```

00214             optimize->file[i][experiment]);
00215     else
00216     {
00217         buffer2 = input->experiment[experiment].stencil[i];
00218         snprintf (buffer, 512, CP " %s %s", buffer2, &cinput[i][0]);
00219         if (system (buffer) == -1)
00220             error_message = g_strdup (buffer);
00221     }
00222     flags <= 1;
00223 }
00224 for (; i < MAX_NINPUTS; ++i)
00225     strcpy (&cinput[i][0], "");
00226 #if DEBUG_OPTIMIZE
00227 fprintf (stderr, "optimize_parse: parsing end\n");
00228 #endif
00229
00230 // Performing the simulation
00231 snprintf (output, 32, "output-%u-%u", simulation, experiment);
00232 buffer2 = g_path_get_dirname (optimize->simulator);
00233 buffer3 = g_path_get_basename (optimize->simulator);
00234 buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235 snprintf (buffer, 512, "%s\\%s\\%s %s %s %s %s %s %s %s",
00236           buffer4, cinput[0], cinput[1], cinput[2], cinput[3], cinput[4],
00237           cinput[5], cinput[6], cinput[7], output);
00238 g_free (buffer4);
00239 g_free (buffer3);
00240 g_free (buffer2);
00241 #if DEBUG_OPTIMIZE
00242 fprintf (stderr, "optimize_parse: %s\n", buffer);
00243 #endif
00244 if (system (buffer) == -1)
00245     error_message = g_strdup (buffer);
00246
00247 // Checking the objective value function
00248 if (optimize->evaluator)
00249 {
00250     snprintf (result, 32, "result-%u-%u", simulation, experiment);
00251     buffer2 = g_path_get_dirname (optimize->evaluator);
00252     buffer3 = g_path_get_basename (optimize->evaluator);
00253     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00254     snprintf (buffer, 512, "%s\\%s\\%s %s %s",
00255             buffer4, output, optimize->experiment[experiment], result);
00256     g_free (buffer4);
00257     g_free (buffer3);
00258     g_free (buffer2);
00259 #if DEBUG_OPTIMIZE
00260 fprintf (stderr, "optimize_parse: %s\n", buffer);
00261 fprintf (stderr, "optimize_parse: result=%s\n", result);
00262 #endif
00263     if (system (buffer) == -1)
00264         error_message = g_strdup (buffer);
00265     file_result = g_fopen (result, "r");
00266     e = atof (fgets (buffer, 512, file_result));
00267     fclose (file_result);
00268 }
00269 else
00270 {
00271 #if DEBUG_OPTIMIZE
00272 fprintf (stderr, "optimize_parse: output=%s\n", output);
00273 #endif
00274     strcpy (result, "");
00275     file_result = g_fopen (output, "r");
00276     e = atof (fgets (buffer, 512, file_result));
00277     fclose (file_result);
00278 }
00279
00280 // Removing files
00281 if (optimize->cleaner)
00282 {
00283     buffer2 = g_path_get_dirname (optimize->cleaner);
00284     buffer3 = g_path_get_basename (optimize->cleaner);
00285     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00286     snprintf (buffer, 512, "%s\\%s\\", buffer4);
00287     g_free (buffer4);
00288     g_free (buffer3);
00289     g_free (buffer2);
00290     if (system (buffer) == -1)
00291         error_message = g_strdup (buffer);
00292 }
00293 #if !DEBUG_OPTIMIZE
00294 for (i = 0; i < optimize->ninputs; ++i)
00295 {
00296     if (optimize->file[i][0])
00297     {
00298         snprintf (buffer, 512, RM " %s", &cinput[i][0]);
00299         if (system (buffer) == -1)
00300             error_message = g_strdup (buffer);
00301     }
00302 }

```

```

00301     }
00302     }
00303     snprintf (buffer, 512, RM " %s %s", output, result);
00304     if (system (buffer) == -1)
00305         error_message = g_strdup (buffer);
00306 #endif
00307
00308     // Processing pending events
00309     if (show_pending)
00310         show_pending ();
00311
00312 #if DEBUG_OPTIMIZE
00313     fprintf (stderr, "optimize_parse: end\n");
00314 #endif
00315
00316     // Returning the objective function
00317     return e * optimize->weight[experiment];
00318 }

```

Here is the call graph for this function:



#### 4.21.3.24 optimize\_print()

```
static void optimize_print ( ) [static]
```

Function to print the results.

Definition at line 427 of file [optimize.c](#).

```

00428 {
00429     unsigned int i;
00430     char buffer[512];
00431 #if HAVE_MPI
00432     if (optimize->mpi_rank)
00433         return;
00434 #endif
00435     printf ("%s\n", _("Best result"));
00436     fprintf (optimize->file_result, "%s\n", _("Best result"));
00437     printf ("error = %.15le\n", optimize->error_old[0]);
00438     fprintf (optimize->file_result, "error = %.15le\n", optimize->error_old[0]);
00439     for (i = 0; i < optimize->nvariables; ++i)
00440     {
00441         snprintf (buffer, 512, "%s = %s\n",
00442                 optimize->label[i], format[optimize->precision[i]]);
00443         printf (buffer, optimize->value_old[i]);
00444         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00445     }
00446     fflush (optimize->file_result);
00447 }

```

#### 4.21.3.25 optimize\_refine()

```
static void optimize_refine ( ) [inline], [static]
```

Function to refine the search ranges of the variables in iterative algorithms.

Definition at line 1318 of file [optimize.c](#).

```

01319 {
01320     unsigned int i, j;
01321     double d;
01322     #if HAVE_MPI
01323     MPI_Status mpi_stat;
01324     #endif
01325     #if DEBUG_OPTIMIZE
01326     fprintf (stderr, "optimize_refine: start\n");
01327     #endif
01328     #if HAVE_MPI
01329     if (!optimize->mpi_rank)
01330     {
01331     #endif
01332         for (j = 0; j < optimize->nvariables; ++j)
01333         {
01334             optimize->rangemin[j] = optimize->rangemax[j]
01335             = optimize->value_old[j];
01336             optimize->step[j] = input->variable[j].step;
01337         }
01338         for (i = 0; ++i < optimize->nbest;)
01339         {
01340             for (j = 0; j < optimize->nvariables; ++j)
01341             {
01342                 optimize->rangemin[j]
01343                 = fmin (optimize->rangemin[j],
01344                     optimize->value_old[i * optimize->nvariables + j]);
01345                 optimize->rangemax[j]
01346                 = fmax (optimize->rangemax[j],
01347                     optimize->value_old[i * optimize->nvariables + j]);
01348             }
01349         }
01350         for (j = 0; j < optimize->nvariables; ++j)
01351         {
01352             d = optimize->tolerance
01353             * (optimize->rangemax[j] - optimize->rangemin[j]);
01354             switch (optimize->algorithm)
01355             {
01356             case ALGORITHM_MONTE_CARLO:
01357                 d *= 0.5;
01358                 break;
01359             default:
01360                 if (optimize->nsweeps[j] > 1)
01361                     d /= optimize->nsweeps[j] - 1;
01362                 else
01363                     d = 0.;
01364             }
01365             optimize->rangemin[j] -= d;
01366             optimize->rangemin[j]
01367             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01368             optimize->rangemax[j] += d;
01369             optimize->rangemax[j]
01370             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01371             printf ("%s min=%lg max=%lg\n", optimize->label[j],
01372                 optimize->rangemin[j], optimize->rangemax[j]);
01373             fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01374                 optimize->label[j], optimize->rangemin[j],
01375                 optimize->rangemax[j]);
01376         }
01377     #if HAVE_MPI
01378     for (i = 1; (int) i < ntasks; ++i)
01379     {
01380         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01381             1, MPI_COMM_WORLD);
01382         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01383             1, MPI_COMM_WORLD);
01384     }
01385     }
01386     else
01387     {
01388         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01389             MPI_COMM_WORLD, &mpi_stat);
01390         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01391             MPI_COMM_WORLD, &mpi_stat);
01392     }
01393     #endif
01394     #if DEBUG_OPTIMIZE
01395     fprintf (stderr, "optimize_refine: end\n");
01396     #endif
01397 }

```

#### 4.21.3.26 optimize\_save\_old()

```
static void optimize_save_old ( ) [inline], [static]
```

Function to save the best results on iterative methods.

Definition at line 1238 of file `optimize.c`.

```

01239 {
01240     unsigned int i, j;
01241     #if DEBUG_OPTIMIZE
01242         fprintf (stderr, "optimize_save_old: start\n");
01243         fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01244     #endif
01245     memcpy (optimize->error_old, optimize->error_best,
01246             optimize->nbest * sizeof (double));
01247     for (i = 0; i < optimize->nbest; ++i)
01248     {
01249         j = optimize->simulation_best[i];
01250     #if DEBUG_OPTIMIZE
01251         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01252     #endif
01253     }
01254     memcpy (optimize->value_old + i * optimize->nvariables,
01255             optimize->value + j * optimize->nvariables,
01256             optimize->nvariables * sizeof (double));
01257     #if DEBUG_OPTIMIZE
01258     for (i = 0; i < optimize->nvariables; ++i)
01259         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01260                 i, optimize->value_old[i]);
01261     fprintf (stderr, "optimize_save_old: end\n");
01262     #endif
01263 }
```

#### 4.21.3.27 optimize\_save\_optimal()

```
static void optimize_save_optimal ( ) [inline], [static]
```

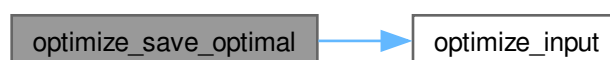
Function to save the optimal input files.

Definition at line 1457 of file `optimize.c`.

```

01458 {
01459     char cinput[32];
01460     unsigned int i, j;
01461     unsigned int flags = 1;
01462     // Getting optimal values
01463     memcpy (optimize->value, optimize->value_old,
01464             optimize->nvariables * sizeof (double));
01465     // Saving optimal input files
01466     for (i = 0; i < optimize->ninputs; ++i)
01467     {
01468         for (j = 0; j < optimize->nexperiments; ++j)
01469         {
01470             snprintf (cinput, 32, "optimal-%u-%u", i, j);
01471         #if DEBUG_OPTIMIZE
01472             fprintf (stderr, "optimize_save_optimal: i=%u j=%u input=%s\n",
01473                     i, j, cinput);
01474         #endif
01475         // Checking templates
01476         if (optimize->template_flags & flags)
01477             optimize_input (0, cinput, optimize->file[i][j]);
01478         flags <= 1;
01479     }
01480 }
```

Here is the call graph for this function:



#### 4.21.3.28 optimize\_save\_variables()

```
static void optimize_save_variables (
    unsigned int simulation,
    double error ) [static]
```

Function to save in a file the variables and the error.

##### Parameters

<i>simulation</i>	Simulation number.
<i>error</i>	Error value.

Definition at line 453 of file [optimize.c](#).

```
00455 {
00456     unsigned int i;
00457     char buffer[64];
00458     #if DEBUG_OPTIMIZE
00459     fprintf (stderr, "optimize_save_variables: start\n");
00460     #endif
00461     for (i = 0; i < optimize->nvariables; ++i)
00462     {
00463         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00464         fprintf (optimize->file_variables, buffer,
00465                 optimize->value[simulation * optimize->nvariables + i]);
00466     }
00467     fprintf (optimize->file_variables, "%.14le\n", error);
00468     fflush (optimize->file_variables);
00469     #if DEBUG_OPTIMIZE
00470     fprintf (stderr, "optimize_save_variables: end\n");
00471     #endif
00472 }
```

#### 4.21.3.29 optimize\_sequential()

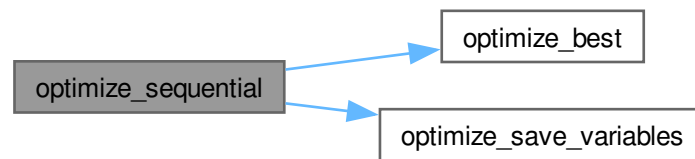
```
static void optimize_sequential ( ) [static]
```

Function to optimize sequentially.

Definition at line 519 of file [optimize.c](#).

```
00520 {
00521     unsigned int i;
00522     double e;
00523     #if DEBUG_OPTIMIZE
00524     fprintf (stderr, "optimize_sequential: start\n");
00525     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00526             optimize->nstart, optimize->nend);
00527     #endif
00528     for (i = optimize->nstart; i < optimize->nend; ++i)
00529     {
00530         e = optimize_norm (i);
00531         optimize_best (i, e);
00532         optimize_save_variables (i, e);
00533         if (e < optimize->threshold)
00534         {
00535             optimize->stop = 1;
00536             break;
00537         }
00538     }
00539     #if DEBUG_OPTIMIZE
00540     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00541     #endif
00542     #if DEBUG_OPTIMIZE
00543     fprintf (stderr, "optimize_sequential: end\n");
00544     #endif
00545 }
```

Here is the call graph for this function:



#### 4.21.3.30 optimize\_step()

```
static void optimize_step ( ) [static]
```

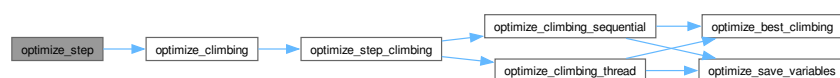
Function to do a step of the iterative algorithm.

Definition at line 1403 of file `optimize.c`.

```

01404 {
01405     #if DEBUG_OPTIMIZE
01406         fprintf (stderr, "optimize_step: start\n");
01407     #endif
01408     optimize_algorithm ();
01409     if (optimize->nsteps)
01410         optimize_climbing (optimize->nsteps);
01411     #if DEBUG_OPTIMIZE
01412         fprintf (stderr, "optimize_step: end\n");
01413     #endif
01414 }
```

Here is the call graph for this function:



#### 4.21.3.31 optimize\_step\_climbing()

```
static void optimize_step_climbing (
    unsigned int simulation ) [inline], [static]
```

Function to do a step of the hill climbing method.

##### Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

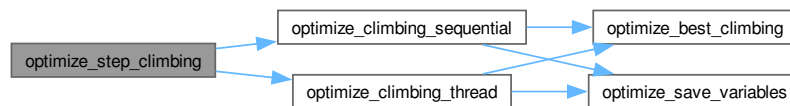
Definition at line 1006 of file `optimize.c`.

```

01007 {
01008     GThread *thread[nthreads_climbing];
01009     ParallelData data[nthreads_climbing];
01010     unsigned int i, j, k, b;
01011     #if DEBUG_OPTIMIZE
01012     fprintf (stderr, "optimize_step_climbing: start\n");
01013     #endif
01014     for (i = 0; i < optimize->nestimates; ++i)
01015     {
01016         k = (simulation + i) * optimize->nvariables;
01017         b = optimize->simulation_best[0] * optimize->nvariables;
01018         #if DEBUG_OPTIMIZE
01019         fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
01020                 simulation + i, optimize->simulation_best[0]);
01021         #endif
01022         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
01023         {
01024             #if DEBUG_OPTIMIZE
01025             fprintf (stderr,
01026                     "optimize_step_climbing: estimate=%u best=%u=%.14le\n",
01027                     i, j, optimize->value[b]);
01028             #endif
01029             optimize->value[k]
01030             = optimize->value[b] + optimize_estimate_climbing (j, i);
01031             optimize->value[k] = fmin (fmax (optimize->value[k],
01032                                             optimize->rangeminabs[j]),
01033                                       optimize->rangemaxabs[j]);
01034             #if DEBUG_OPTIMIZE
01035             fprintf (stderr,
01036                     "optimize_step_climbing: estimate=%u variable=%u=%.14le\n",
01037                     i, j, optimize->value[k]);
01038             #endif
01039         }
01040     }
01041     if (nthreads_climbing == 1)
01042         optimize_climbing_sequential (simulation);
01043     else
01044     {
01045         for (i = 0; i <= nthreads_climbing; ++i)
01046         {
01047             optimize->thread_climbing[i]
01048             = simulation + optimize->nstart_climbing
01049             + i * (optimize->nend_climbing - optimize->nstart_climbing)
01050             / nthreads_climbing;
01051             #if DEBUG_OPTIMIZE
01052             fprintf (stderr,
01053                     "optimize_step_climbing: i=%u thread_climbing=%u\n",
01054                     i, optimize->thread_climbing[i]);
01055             #endif
01056         }
01057         for (i = 0; i < nthreads_climbing; ++i)
01058         {
01059             data[i].thread = i;
01060             thread[i] = g_thread_new
01061             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01062         }
01063         for (i = 0; i < nthreads_climbing; ++i)
01064             g_thread_join (thread[i]);
01065     }
01066     #if DEBUG_OPTIMIZE
01067     fprintf (stderr, "optimize_step_climbing: end\n");
01068     #endif
01069 }

```

Here is the call graph for this function:



#### 4.21.3.32 optimize\_sweep()

```
static void optimize_sweep ( ) [static]
```



Function to optimize with the sweep algorithm.

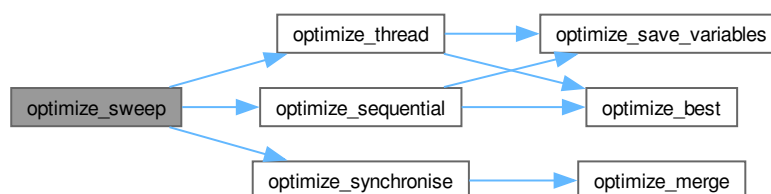
Definition at line 699 of file `optimize.c`.

```

00700 {
00701     ParallelData data[nthreads];
00702     GThread *thread[nthreads];
00703     double range[optimize->nvariables];
00704     double e;
00705     unsigned int i, j, k, l;
00706     #if DEBUG_OPTIMIZE
00707     fprintf (stderr, "optimize_sweep: start\n");
00708     #endif
00709     for (j = 0; j < optimize->nvariables; ++j)
00710         range[j] = (optimize->rangemax[j] - optimize->rangemin[j])
00711             / (optimize->nsweeps[j] - 1);
00712     for (i = 0; i < optimize->nsimulations; ++i)
00713     {
00714         k = i;
00715         for (j = 0; j < optimize->nvariables; ++j)
00716         {
00717             l = k % optimize->nsweeps[j];
00718             k /= optimize->nsweeps[j];
00719             e = optimize->rangemin[j];
00720             if (optimize->nsweeps[j] > 1)
00721                 e += l * range[j];
00722             optimize->value[i * optimize->nvariables + j] = e;
00723         }
00724     }
00725     optimize->nsaveds = 0;
00726     if (nthreads <= 1)
00727         optimize_sequential ();
00728     else
00729     {
00730         for (i = 0; i < nthreads; ++i)
00731         {
00732             data[i].thread = i;
00733             thread[i]
00734                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00735         }
00736         for (i = 0; i < nthreads; ++i)
00737             g_thread_join (thread[i]);
00738     }
00739     #if HAVE_MPI
00740     // Communicating tasks results
00741     optimize_synchronise ();
00742     #endif
00743     #if DEBUG_OPTIMIZE
00744     fprintf (stderr, "optimize_sweep: end\n");
00745     #endif
00746 }

```

Here is the call graph for this function:



#### 4.21.3.33 optimize\_synchronise()

```
static void optimize_synchronise ( ) [static]
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 652 of file `optimize.c`.

```

00653 {
00654     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00655     double error_best[optimize->nbest];
00656     MPI_Status mpi_stat;
00657     #if DEBUG_OPTIMIZE
00658     fprintf(stderr, "optimize_synchronise: start\n");
00659     #endif
00660     if (optimize->mpi_rank == 0)
00661     {
00662         for (i = 1; (int) i < ntasks; ++i)
00663         {
00664             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00665             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00666                     MPI_COMM_WORLD, &mpi_stat);
00667             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00668                     MPI_COMM_WORLD, &mpi_stat);
00669             optimize_merge (nsaveds, simulation_best, error_best);
00670             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00671             if (stop)
00672                 optimize->stop = 1;
00673         }
00674         for (i = 1; (int) i < ntasks; ++i)
00675             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00676     }
00677     else
00678     {
00679         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00680         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00681                 MPI_COMM_WORLD);
00682         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00683                 MPI_COMM_WORLD);
00684         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00685         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00686         if (stop)
00687             optimize->stop = 1;
00688     }
00689     #if DEBUG_OPTIMIZE
00690     fprintf(stderr, "optimize_synchronise: end\n");
00691     #endif
00692 }

```

Here is the call graph for this function:



#### 4.21.3.34 optimize\_thread()

```

static void * optimize_thread (
    ParallelData * data ) [static]

```

Function to optimize on a thread.

##### Returns

NULL.

##### Parameters

<i>data</i>	Function data.
-------------	----------------

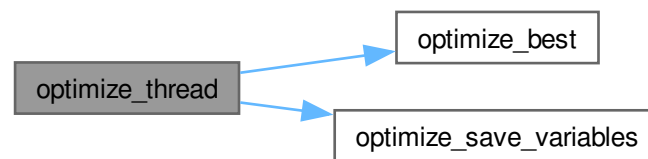
Definition at line 553 of file [optimize.c](#).

```

00554 {
00555     unsigned int i, thread;
00556     double e;
00557     #if DEBUG_OPTIMIZE
00558     fprintf (stderr, "optimize_thread: start\n");
00559     #endif
00560     thread = data->thread;
00561     #if DEBUG_OPTIMIZE
00562     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00563             optimize->thread[thread], optimize->thread[thread + 1]);
00564     #endif
00565     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00566     {
00567         e = optimize_norm (i);
00568         g_mutex_lock (mutex);
00569         optimize_best (i, e);
00570         optimize_save_variables (i, e);
00571         if (e < optimize->threshold)
00572             optimize->stop = 1;
00573         g_mutex_unlock (mutex);
00574         if (optimize->stop)
00575             break;
00576     #if DEBUG_OPTIMIZE
00577     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00578     #endif
00579     }
00580     #if DEBUG_OPTIMIZE
00581     fprintf (stderr, "optimize_thread: end\n");
00582     #endif
00583     g_thread_exit (NULL);
00584     return NULL;
00585 }

```

Here is the call graph for this function:



## 4.21.4 Variable Documentation

### 4.21.4.1 nthreads\_climbing

```
unsigned int nthreads_climbing
```

Number of threads for the hill climbing method.

Definition at line 84 of file [optimize.c](#).

### 4.21.4.2 optimize

```
Optimize optimize[1]
```

Optimization data.

Definition at line 83 of file [optimize.c](#).

#### 4.21.4.3 optimize\_algorithm

```
void(* optimize_algorithm) () ( ) [static]
```

Pointer to the function to perform a optimization algorithm step.

Definition at line 87 of file [optimize.c](#).

#### 4.21.4.4 optimize\_estimate\_climbing

```
double(* optimize_estimate_climbing) (unsigned int variable, unsigned int estimate) (
    unsigned int variable,
    unsigned int estimate ) [static]
```

Pointer to the function to estimate the climbing.

Definition at line 89 of file [optimize.c](#).

#### 4.21.4.5 optimize\_norm

```
double(* optimize_norm) (unsigned int simulation) (
    unsigned int simulation ) [static]
```

Pointer to the error norm function.

Definition at line 92 of file [optimize.c](#).

## 4.22 optimize.c

[Go to the documentation of this file.](#)

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS" AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
```

```

00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <sys/param.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <glib/gstdio.h>
00050 #include <json-glib/json-glib.h>
00051 #ifdef G_OS_WIN32
00052 #include <windows.h>
00053 #elif !defined(__BSD_VISIBLE) && !defined(NetBSD)
00054 #include <alloca.h>
00055 #endif
00056 #if HAVE_MPI
00057 #include <mpi.h>
00058 #endif
00059 #include "jb/src/jb_win.h"
00060 #include "genetic/genetic.h"
00061 #include "tools.h"
00062 #include "experiment.h"
00063 #include "variable.h"
00064 #include "input.h"
00065 #include "optimize.h"
00066
00067 #define DEBUG_OPTIMIZE 0
00068
00075 #ifdef G_OS_WIN32
00076 #define CP "copy"
00077 #define RM "del"
00078 #else
00079 #define CP "cp"
00080 #define RM "rm"
00081 #endif
00082
00083 Optimize optimize[1];
00084 unsigned int nthreads_climbing;
00086
00087 static void (*optimize_algorithm) ();
00089 static double (*optimize_estimate_climbing) (unsigned int variable,
00090                                             unsigned int estimate);
00092 static double (*optimize_norm) (unsigned int simulation);
00094
00098 static inline void
00099 optimize_input (unsigned int simulation,
00100                char *input,
00101                GMappedFile * stencil)
00102 {
00103     char buffer[256], value[32];
00104     GRegex *regex;
00105     FILE *file;
00106     char *buffer2, *buffer3 = NULL, *content;
00107     gsize length;
00108     unsigned int i;
00109
00110 #if DEBUG_OPTIMIZE
00111     fprintf (stderr, "optimize_input: start\n");
00112 #endif
00113
00114     // Checking the file
00115     if (!stencil)
00116         goto optimize_input_end;
00117
00118     // Opening stencil
00119     content = g_mapped_file_get_contents (stencil);
00120     length = g_mapped_file_get_length (stencil);
00121 #if DEBUG_OPTIMIZE
00122     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00123 #endif
00124     file = g_fopen (input, "w");
00125
00126     // Parsing stencil
00127     for (i = 0; i < optimize->nvariables; ++i)
00128     {
00129 #if DEBUG_OPTIMIZE
00130         fprintf (stderr, "optimize_input: variable=%u\n", i);
00131 #endif
00132         snprintf (buffer, 32, "@variable%u@", i + 1);
00133         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00134                             NULL);
00135         if (i == 0)
00136         {
00137             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00138                                                optimize->label[i],
00139                                                (GRegexMatchFlags) 0, NULL);

```

```

00140 #if DEBUG_OPTIMIZE
00141     fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00142 #endif
00143 }
00144 else
00145 {
00146     length = strlen (buffer3);
00147     buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00148                                     optimize->label[i],
00149                                     (GRegexMatchFlags) 0, NULL);
00150     g_free (buffer3);
00151 }
00152 g_regex_unref (regex);
00153 length = strlen (buffer2);
00154 snprintf (buffer, 32, "@value%u@", i + 1);
00155 regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00156                     NULL);
00157 snprintf (value, 32, format[optimize->precision[i]],
00158          optimize->value[simulation * optimize->nvariables + i]);
00159
00160 #if DEBUG_OPTIMIZE
00161     fprintf (stderr, "optimize_input: value=%s\n", value);
00162 #endif
00163 buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00164                                  (GRegexMatchFlags) 0, NULL);
00165 g_free (buffer2);
00166 g_regex_unref (regex);
00167 }
00168
00169 // Saving input file
00170 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00171 g_free (buffer3);
00172 fclose (file);
00173
00174 optimize_input_end:
00175 #if DEBUG_OPTIMIZE
00176     fprintf (stderr, "optimize_input: end\n");
00177 #endif
00178     return;
00179 }
00180
00181 static double
00182 optimize_parse (unsigned int simulation,
00183                unsigned int experiment)
00184 {
00185     char buffer[512], cinput[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00186     *buffer3, *buffer4;
00187     FILE *file_result;
00188     double e;
00189     unsigned int i;
00190     unsigned int flags = 1;
00191
00192     #if DEBUG_OPTIMIZE
00193         fprintf (stderr, "optimize_parse: start\n");
00194         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00195                 simulation, experiment);
00196     #endif
00197
00198     // Opening input files
00199     for (i = 0; i < optimize->ninputs; ++i)
00200     {
00201         snprintf (&cinput[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00202         #if DEBUG_OPTIMIZE
00203             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &cinput[i][0]);
00204         #endif
00205         // Checking simple copy
00206         if (optimize->template_flags & flags)
00207             optimize_input (simulation, &cinput[i][0],
00208                             optimize->file[i][experiment]);
00209         else
00210         {
00211             buffer2 = input->experiment[experiment].stencil[i];
00212             snprintf (buffer, 512, CP " %s %s", buffer2, &cinput[i][0]);
00213             if (system (buffer) == -1)
00214                 error_message = g_strdup (buffer);
00215         }
00216         flags <= 1;
00217     }
00218     for (; i < MAX_NINPUTS; ++i)
00219         strcpy (&cinput[i][0], "");
00220     #if DEBUG_OPTIMIZE
00221         fprintf (stderr, "optimize_parse: parsing end\n");
00222     #endif
00223
00224     // Performing the simulation
00225     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00226     buffer2 = g_path_get_dirname (optimize->simulator);

```

```

00233     buffer3 = g_path_get_basename (optimize->simulator);
00234     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s",
00236             buffer4, cinput[0], cinput[1], cinput[2], cinput[3], cinput[4],
00237             cinput[5], cinput[6], cinput[7], output);
00238     g_free (buffer4);
00239     g_free (buffer3);
00240     g_free (buffer2);
00241     #if DEBUG_OPTIMIZE
00242     fprintf (stderr, "optimize_parse: %s\n", buffer);
00243     #endif
00244     if (system (buffer) == -1)
00245         error_message = g_strdup (buffer);
00246
00247     // Checking the objective value function
00248     if (optimize->evaluator)
00249     {
00250         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00251         buffer2 = g_path_get_dirname (optimize->evaluator);
00252         buffer3 = g_path_get_basename (optimize->evaluator);
00253         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00254         snprintf (buffer, 512, "\"%s\" %s %s %s",
00255             buffer4, output, optimize->experiment[experiment], result);
00256         g_free (buffer4);
00257         g_free (buffer3);
00258         g_free (buffer2);
00259         #if DEBUG_OPTIMIZE
00260         fprintf (stderr, "optimize_parse: %s\n", buffer);
00261         fprintf (stderr, "optimize_parse: result=%s\n", result);
00262         #endif
00263         if (system (buffer) == -1)
00264             error_message = g_strdup (buffer);
00265         file_result = g_fopen (result, "r");
00266         e = atof (fgets (buffer, 512, file_result));
00267         fclose (file_result);
00268     }
00269     else
00270     {
00271         #if DEBUG_OPTIMIZE
00272         fprintf (stderr, "optimize_parse: output=%s\n", output);
00273         #endif
00274         strcpy (result, "");
00275         file_result = g_fopen (output, "r");
00276         e = atof (fgets (buffer, 512, file_result));
00277         fclose (file_result);
00278     }
00279
00280     // Removing files
00281     if (optimize->cleaner)
00282     {
00283         buffer2 = g_path_get_dirname (optimize->cleaner);
00284         buffer3 = g_path_get_basename (optimize->cleaner);
00285         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00286         snprintf (buffer, 512, "\"%s\"", buffer4);
00287         g_free (buffer4);
00288         g_free (buffer3);
00289         g_free (buffer2);
00290         if (system (buffer) == -1)
00291             error_message = g_strdup (buffer);
00292     }
00293     #if !DEBUG_OPTIMIZE
00294     for (i = 0; i < optimize->ninputs; ++i)
00295     {
00296         if (optimize->file[i][0])
00297         {
00298             snprintf (buffer, 512, RM " %s", &cinput[i][0]);
00299             if (system (buffer) == -1)
00300                 error_message = g_strdup (buffer);
00301         }
00302     }
00303     snprintf (buffer, 512, RM " %s %s", output, result);
00304     if (system (buffer) == -1)
00305         error_message = g_strdup (buffer);
00306     #endif
00307
00308     // Processing pending events
00309     if (show_pending)
00310         show_pending ();
00311
00312     #if DEBUG_OPTIMIZE
00313     fprintf (stderr, "optimize_parse: end\n");
00314     #endif
00315
00316     // Returning the objective function
00317     return e * optimize->weight[experiment];
00318 }
00319

```

```

00325 static double
00326 optimize_norm_euclidian (unsigned int simulation)
00327 {
00328     double e, ei;
00329     unsigned int i;
00330     #if DEBUG_OPTIMIZE
00331     fprintf (stderr, "optimize_norm_euclidian: start\n");
00332     #endif
00333     e = 0.;
00334     for (i = 0; i < optimize->nexperiments; ++i)
00335     {
00336         ei = optimize_parse (simulation, i);
00337         e += ei * ei;
00338     }
00339     e = sqrt (e);
00340     #if DEBUG_OPTIMIZE
00341     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00342     fprintf (stderr, "optimize_norm_euclidian: end\n");
00343     #endif
00344     return e;
00345 }
00346
00352 static double
00353 optimize_norm_maximum (unsigned int simulation)
00354 {
00355     double e, ei;
00356     unsigned int i;
00357     #if DEBUG_OPTIMIZE
00358     fprintf (stderr, "optimize_norm_maximum: start\n");
00359     #endif
00360     e = 0.;
00361     for (i = 0; i < optimize->nexperiments; ++i)
00362     {
00363         ei = fabs (optimize_parse (simulation, i));
00364         e = fmax (e, ei);
00365     }
00366     #if DEBUG_OPTIMIZE
00367     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00368     fprintf (stderr, "optimize_norm_maximum: end\n");
00369     #endif
00370     return e;
00371 }
00372
00378 static double
00379 optimize_norm_p (unsigned int simulation)
00380 {
00381     double e, ei;
00382     unsigned int i;
00383     #if DEBUG_OPTIMIZE
00384     fprintf (stderr, "optimize_norm_p: start\n");
00385     #endif
00386     e = 0.;
00387     for (i = 0; i < optimize->nexperiments; ++i)
00388     {
00389         ei = fabs (optimize_parse (simulation, i));
00390         e += pow (ei, optimize->p);
00391     }
00392     e = pow (e, 1. / optimize->p);
00393     #if DEBUG_OPTIMIZE
00394     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00395     fprintf (stderr, "optimize_norm_p: end\n");
00396     #endif
00397     return e;
00398 }
00399
00405 static double
00406 optimize_norm_taxicab (unsigned int simulation)
00407 {
00408     double e;
00409     unsigned int i;
00410     #if DEBUG_OPTIMIZE
00411     fprintf (stderr, "optimize_norm_taxicab: start\n");
00412     #endif
00413     e = 0.;
00414     for (i = 0; i < optimize->nexperiments; ++i)
00415         e += fabs (optimize_parse (simulation, i));
00416     #if DEBUG_OPTIMIZE
00417     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00418     fprintf (stderr, "optimize_norm_taxicab: end\n");
00419     #endif
00420     return e;
00421 }
00422
00426 static void
00427 optimize_print ()
00428 {
00429     unsigned int i;

```



```

00430     char buffer[512];
00431     #if HAVE_MPI
00432     if (optimize->mpi_rank)
00433         return;
00434     #endif
00435     printf ("%s\n", _("Best result"));
00436     fprintf (optimize->file_result, "%s\n", _("Best result"));
00437     printf ("error = %.15le\n", optimize->error_old[0]);
00438     fprintf (optimize->file_result, "error = %.15le\n", optimize->error_old[0]);
00439     for (i = 0; i < optimize->nvariables; ++i)
00440     {
00441         snprintf (buffer, 512, "%s = %s\n",
00442                 optimize->label[i], format[optimize->precision[i]]);
00443         printf (buffer, optimize->value_old[i]);
00444         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00445     }
00446     fflush (optimize->file_result);
00447 }
00448
00452 static void
00453 optimize_save_variables (unsigned int simulation,
00454                         double error)
00455 {
00456     unsigned int i;
00457     char buffer[64];
00458     #if DEBUG_OPTIMIZE
00459     fprintf (stderr, "optimize_save_variables: start\n");
00460     #endif
00461     for (i = 0; i < optimize->nvariables; ++i)
00462     {
00463         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00464         fprintf (optimize->file_variables, buffer,
00465                 optimize->value[simulation * optimize->nvariables + i]);
00466     }
00467     fprintf (optimize->file_variables, "%.14le\n", error);
00468     fflush (optimize->file_variables);
00469     #if DEBUG_OPTIMIZE
00470     fprintf (stderr, "optimize_save_variables: end\n");
00471     #endif
00472 }
00473
00477 static void
00478 optimize_best (unsigned int simulation,
00479               double value)
00480 {
00481     unsigned int i, j;
00482     double e;
00483     #if DEBUG_OPTIMIZE
00484     fprintf (stderr, "optimize_best: start\n");
00485     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00486             optimize->nsaveds, optimize->nbest);
00487     #endif
00488     if (optimize->nsaveds < optimize->nbest
00489         || value < optimize->error_best[optimize->nsaveds - 1])
00490     {
00491         if (optimize->nsaveds < optimize->nbest)
00492             ++optimize->nsaveds;
00493         optimize->error_best[optimize->nsaveds - 1] = value;
00494         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00495         for (i = optimize->nsaveds; --i;)
00496         {
00497             if (optimize->error_best[i] < optimize->error_best[i - 1])
00498             {
00499                 j = optimize->simulation_best[i];
00500                 e = optimize->error_best[i];
00501                 optimize->simulation_best[i] = optimize->simulation_best[i - 1];
00502                 optimize->error_best[i] = optimize->error_best[i - 1];
00503                 optimize->simulation_best[i - 1] = j;
00504                 optimize->error_best[i - 1] = e;
00505             }
00506             else
00507                 break;
00508         }
00509     }
00510     #if DEBUG_OPTIMIZE
00511     fprintf (stderr, "optimize_best: end\n");
00512     #endif
00513 }
00514
00518 static void
00519 optimize_sequential ()
00520 {
00521     unsigned int i;
00522     double e;
00523     #if DEBUG_OPTIMIZE
00524     fprintf (stderr, "optimize_sequential: start\n");
00525     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",

```

```

00526         optimize->nstart, optimize->nend);
00527 #endif
00528     for (i = optimize->nstart; i < optimize->nend; ++i)
00529     {
00530         e = optimize_norm (i);
00531         optimize_best (i, e);
00532         optimize_save_variables (i, e);
00533         if (e < optimize->threshold)
00534         {
00535             optimize->stop = 1;
00536             break;
00537         }
00538 #if DEBUG_OPTIMIZE
00539         fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00540 #endif
00541     }
00542 #if DEBUG_OPTIMIZE
00543     fprintf (stderr, "optimize_sequential: end\n");
00544 #endif
00545 }
00546
00552 static void *
00553 optimize_thread (ParallelData * data)
00554 {
00555     unsigned int i, thread;
00556     double e;
00557 #if DEBUG_OPTIMIZE
00558     fprintf (stderr, "optimize_thread: start\n");
00559 #endif
00560     thread = data->thread;
00561 #if DEBUG_OPTIMIZE
00562     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00563             optimize->thread[thread], optimize->thread[thread + 1]);
00564 #endif
00565     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00566     {
00567         e = optimize_norm (i);
00568         g_mutex_lock (mutex);
00569         optimize_best (i, e);
00570         optimize_save_variables (i, e);
00571         if (e < optimize->threshold)
00572             optimize->stop = 1;
00573         g_mutex_unlock (mutex);
00574         if (optimize->stop)
00575             break;
00576 #if DEBUG_OPTIMIZE
00577         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00578 #endif
00579     }
00580 #if DEBUG_OPTIMIZE
00581     fprintf (stderr, "optimize_thread: end\n");
00582 #endif
00583     g_thread_exit (NULL);
00584     return NULL;
00585 }
00586
00590 static inline void
00591 optimize_merge (unsigned int nsaveds,
00592                unsigned int *simulation_best,
00593                double *error_best)
00594 {
00595     unsigned int i, j, k, s[optimize->nbest];
00596     double e[optimize->nbest];
00597 #if DEBUG_OPTIMIZE
00598     fprintf (stderr, "optimize_merge: start\n");
00599 #endif
00600 #endif
00601     i = j = k = 0;
00602     do
00603     {
00604         if (i == optimize->nsaveds)
00605         {
00606             s[k] = simulation_best[j];
00607             e[k] = error_best[j];
00608             ++j;
00609             ++k;
00610             if (j == nsaveds)
00611                 break;
00612         }
00613     }
00614     else if (j == nsaveds)
00615     {
00616         s[k] = optimize->simulation_best[i];
00617         e[k] = optimize->error_best[i];
00618         ++i;
00619         ++k;
00620         if (i == optimize->nsaveds)
00621             break;
00622     }

```

```

00623     else if (optimize->error_best[i] > error_best[j])
00624     {
00625         s[k] = simulation_best[j];
00626         e[k] = error_best[j];
00627         ++j;
00628         ++k;
00629     }
00630     else
00631     {
00632         s[k] = optimize->simulation_best[i];
00633         e[k] = optimize->error_best[i];
00634         ++i;
00635         ++k;
00636     }
00637 }
00638 while (k < optimize->nbest);
00639 optimize->nsaveds = k;
00640 memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00641 memcpy (optimize->error_best, e, k * sizeof (double));
00642 #if DEBUG_OPTIMIZE
00643     fprintf (stderr, "optimize_merge: end\n");
00644 #endif
00645 }
00646
00650 #if HAVE_MPI
00651 static void
00652 optimize_synchronise ()
00653 {
00654     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00655     double error_best[optimize->nbest];
00656     MPI_Status mpi_stat;
00657     #if DEBUG_OPTIMIZE
00658         fprintf (stderr, "optimize_synchronise: start\n");
00659     #endif
00660     if (optimize->mpi_rank == 0)
00661     {
00662         for (i = 1; (int) i < ntasks; ++i)
00663         {
00664             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00665             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00666                     MPI_COMM_WORLD, &mpi_stat);
00667             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00668                     MPI_COMM_WORLD, &mpi_stat);
00669             optimize_merge (nsaveds, simulation_best, error_best);
00670             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00671             if (stop)
00672                 optimize->stop = 1;
00673         }
00674         for (i = 1; (int) i < ntasks; ++i)
00675             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00676     }
00677     else
00678     {
00679         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00680         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00681                 MPI_COMM_WORLD);
00682         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00683                 MPI_COMM_WORLD);
00684         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00685         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00686         if (stop)
00687             optimize->stop = 1;
00688     }
00689     #if DEBUG_OPTIMIZE
00690         fprintf (stderr, "optimize_synchronise: end\n");
00691     #endif
00692 }
00693 #endif
00694
00698 static void
00699 optimize_sweep ()
00700 {
00701     ParallelData data[nthreads];
00702     GThread *thread[nthreads];
00703     double range[optimize->nvariables];
00704     double e;
00705     unsigned int i, j, k, l;
00706     #if DEBUG_OPTIMIZE
00707         fprintf (stderr, "optimize_sweep: start\n");
00708     #endif
00709     for (j = 0; j < optimize->nvariables; ++j)
00710         range[j] = (optimize->rangemax[j] - optimize->rangemin[j])
00711             / (optimize->nsweeps[j] - 1);
00712     for (i = 0; i < optimize->nsimulations; ++i)
00713     {
00714         k = i;
00715         for (j = 0; j < optimize->nvariables; ++j)

```

```

00716     {
00717         l = k % optimize->nsweeps[j];
00718         k /= optimize->nsweeps[j];
00719         e = optimize->rangemin[j];
00720         if (optimize->nsweeps[j] > 1)
00721             e += 1 * range[j];
00722         optimize->value[i * optimize->nvariables + j] = e;
00723     }
00724 }
00725 optimize->nsaveds = 0;
00726 if (nthreads <= 1)
00727     optimize_sequential ();
00728 else
00729     {
00730         for (i = 0; i < nthreads; ++i)
00731         {
00732             data[i].thread = i;
00733             thread[i]
00734                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00735         }
00736         for (i = 0; i < nthreads; ++i)
00737             g_thread_join (thread[i]);
00738     }
00739 #if HAVE_MPI
00740     // Communicating tasks results
00741     optimize_synchronise ();
00742 #endif
00743 #if DEBUG_OPTIMIZE
00744     fprintf (stderr, "optimize_sweep: end\n");
00745 #endif
00746 }
00747
00751 static void
00752 optimize_MonteCarlo ()
00753 {
00754     ParallelData data[nthreads];
00755     GThread *thread[nthreads];
00756     double range[optimize->nvariables];
00757     unsigned int i, j;
00758 #if DEBUG_OPTIMIZE
00759     fprintf (stderr, "optimize_MonteCarlo: start\n");
00760 #endif
00761     for (j = 0; j < optimize->nvariables; ++j)
00762         range[j] = optimize->rangemax[j] - optimize->rangemin[j];
00763     for (i = 0; i < optimize->nsimulations; ++i)
00764         for (j = 0; j < optimize->nvariables; ++j)
00765             optimize->value[i * optimize->nvariables + j]
00766                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng) * range[j];
00767     optimize->nsaveds = 0;
00768     if (nthreads <= 1)
00769         optimize_sequential ();
00770     else
00771     {
00772         for (i = 0; i < nthreads; ++i)
00773         {
00774             data[i].thread = i;
00775             thread[i]
00776                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00777         }
00778         for (i = 0; i < nthreads; ++i)
00779             g_thread_join (thread[i]);
00780     }
00781 #if HAVE_MPI
00782     // Communicating tasks results
00783     optimize_synchronise ();
00784 #endif
00785 #if DEBUG_OPTIMIZE
00786     fprintf (stderr, "optimize_MonteCarlo: end\n");
00787 #endif
00788 }
00789
00793 static void
00794 optimize_orthogonal ()
00795 {
00796     ParallelData data[nthreads];
00797     GThread *thread[nthreads];
00798     double range[optimize->nvariables];
00799     double e;
00800     unsigned int i, j, k, l;
00801 #if DEBUG_OPTIMIZE
00802     fprintf (stderr, "optimize_orthogonal: start\n");
00803 #endif
00804     for (j = 0; j < optimize->nvariables; ++j)
00805         range[j] = (optimize->rangemax[j] - optimize->rangemin[j])
00806             / optimize->nsweeps[j];
00807     for (i = 0; i < optimize->nsimulations; ++i)
00808     {

```

```

00809     k = i;
00810     for (j = 0; j < optimize->nvariables; ++j)
00811     {
00812         l = k % optimize->nsweeps[j];
00813         k /= optimize->nsweeps[j];
00814         e = optimize->rangemin[j];
00815         if (optimize->nsweeps[j] > 1)
00816             e += (1 + gsl_rng_uniform (optimize->rng)) * range[j];
00817         optimize->value[i * optimize->nvariables + j] = e;
00818     }
00819 }
00820 optimize->nsaveds = 0;
00821 if (nthreads <= 1)
00822     optimize_sequential ();
00823 else
00824 {
00825     for (i = 0; i < nthreads; ++i)
00826     {
00827         data[i].thread = i;
00828         thread[i]
00829             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00830     }
00831     for (i = 0; i < nthreads; ++i)
00832         g_thread_join (thread[i]);
00833 }
00834 #if HAVE_MPI
00835 // Communicating tasks results
00836 optimize_synchronise ();
00837 #endif
00838 #if DEBUG_OPTIMIZE
00839 fprintf (stderr, "optimize_orthogonal: end\n");
00840 #endif
00841 }
00842
00843 static void
00844 optimize_best_climbing (unsigned int simulation,
00845                        double value)
00846 {
00847     #if DEBUG_OPTIMIZE
00848     fprintf (stderr, "optimize_best_climbing: start\n");
00849     fprintf (stderr,
00850             "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00851             simulation, value, optimize->error_best[0]);
00852     #endif
00853     if (value < optimize->error_best[0])
00854     {
00855         optimize->error_best[0] = value;
00856         optimize->simulation_best[0] = simulation;
00857     }
00858     #if DEBUG_OPTIMIZE
00859     fprintf (stderr,
00860             "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00861             simulation, value);
00862     #endif
00863 }
00864 #if DEBUG_OPTIMIZE
00865 fprintf (stderr, "optimize_best_climbing: end\n");
00866 #endif
00867
00868 static inline void
00869 optimize_climbing_sequential (unsigned int simulation)
00870 {
00871     double e;
00872     unsigned int i, j;
00873     #if DEBUG_OPTIMIZE
00874     fprintf (stderr, "optimize_climbing_sequential: start\n");
00875     fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00876             "nend_climbing=%u\n",
00877             optimize->nstart_climbing, optimize->nend_climbing);
00878     #endif
00879     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00880     {
00881         j = simulation + i;
00882         e = optimize_norm (j);
00883         optimize_best_climbing (j, e);
00884         optimize_save_variables (j, e);
00885         if (e < optimize->threshold)
00886         {
00887             optimize->stop = 1;
00888             break;
00889         }
00890     }
00891     #if DEBUG_OPTIMIZE
00892     fprintf (stderr, "optimize_climbing_sequential: i=%u e=%.14le\n", i, e);
00893     #endif
00894 }
00895 #if DEBUG_OPTIMIZE
00896 fprintf (stderr, "optimize_climbing_sequential: end\n");
00897 #endif

```

```

00902 #endif
00903 }
00904
00910 static void *
00911 optimize_climbing_thread (ParallelData * data)
00912 {
00913     unsigned int i, thread;
00914     double e;
00915     #if DEBUG_OPTIMIZE
00916     fprintf (stderr, "optimize_climbing_thread: start\n");
00917     #endif
00918     thread = data->thread;
00919     #if DEBUG_OPTIMIZE
00920     fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00921             thread,
00922             optimize->thread_climbing[thread],
00923             optimize->thread_climbing[thread + 1]);
00924     #endif
00925     for (i = optimize->thread_climbing[thread];
00926          i < optimize->thread_climbing[thread + 1]; ++i)
00927     {
00928         e = optimize_norm (i);
00929         g_mutex_lock (mutex);
00930         optimize_best_climbing (i, e);
00931         optimize_save_variables (i, e);
00932         if (e < optimize->threshold)
00933             optimize->stop = 1;
00934         g_mutex_unlock (mutex);
00935         if (optimize->stop)
00936             break;
00937     #if DEBUG_OPTIMIZE
00938     fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00939     #endif
00940     }
00941     #if DEBUG_OPTIMIZE
00942     fprintf (stderr, "optimize_climbing_thread: end\n");
00943     #endif
00944     g_thread_exit (NULL);
00945     return NULL;
00946 }
00947
00951 static double
00952 optimize_estimate_climbing_random (unsigned int variable,
00953                                     unsigned int estimate
00954                                     __attribute__((unused)))
00955 {
00956     double x;
00957     #if DEBUG_OPTIMIZE
00958     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00959     #endif
00960     x = optimize->climbing[variable]
00961         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00962     #if DEBUG_OPTIMIZE
00963     fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
00964             variable, x);
00965     fprintf (stderr, "optimize_estimate_climbing_random: end\n");
00966     #endif
00967     return x;
00968 }
00971
00975 static double
00976 optimize_estimate_climbing_coordinates (unsigned int variable,
00977                                         unsigned int estimate)
00978 {
00979     double x;
00980     #if DEBUG_OPTIMIZE
00981     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00982     #endif
00983     x = optimize->climbing[variable];
00984     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00985     {
00986         if (estimate & 1)
00987             x += optimize->step[variable];
00988         else
00989             x -= optimize->step[variable];
00990     }
00991     #if DEBUG_OPTIMIZE
00992     fprintf (stderr,
00993             "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
00994             variable, x);
00995     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00996     #endif
00997     return x;
01000 }
01001
01005 static inline void
01006 optimize_step_climbing (unsigned int simulation)

```

```

01007 {
01008     GThread *thread[nthreads_climbing];
01009     ParallelData data[nthreads_climbing];
01010     unsigned int i, j, k, b;
01011     #if DEBUG_OPTIMIZE
01012     fprintf (stderr, "optimize_step_climbing: start\n");
01013     #endif
01014     for (i = 0; i < optimize->nestimates; ++i)
01015     {
01016         k = (simulation + i) * optimize->nvariables;
01017         b = optimize->simulation_best[0] * optimize->nvariables;
01018         #if DEBUG_OPTIMIZE
01019         fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
01020                 simulation + i, optimize->simulation_best[0]);
01021         #endif
01022         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
01023         {
01024             #if DEBUG_OPTIMIZE
01025             fprintf (stderr,
01026                     "optimize_step_climbing: estimate=%u best=%u=%.14le\n",
01027                     i, j, optimize->value[b]);
01028             #endif
01029             optimize->value[k]
01030             = optimize->value[b] + optimize_estimate_climbing (j, i);
01031             optimize->value[k] = fmin (fmax (optimize->value[k],
01032                                             optimize->rangeminabs[j]),
01033                                     optimize->rangemaxabs[j]);
01034             #if DEBUG_OPTIMIZE
01035             fprintf (stderr,
01036                     "optimize_step_climbing: estimate=%u variable=%u=%.14le\n",
01037                     i, j, optimize->value[k]);
01038             #endif
01039         }
01040     }
01041     if (nthreads_climbing == 1)
01042         optimize_climbing_sequential (simulation);
01043     else
01044     {
01045         for (i = 0; i <= nthreads_climbing; ++i)
01046         {
01047             optimize->thread_climbing[i]
01048             = simulation + optimize->nstart_climbing
01049             + i * (optimize->nend_climbing - optimize->nstart_climbing)
01050             / nthreads_climbing;
01051             #if DEBUG_OPTIMIZE
01052             fprintf (stderr,
01053                     "optimize_step_climbing: i=%u thread_climbing=%u\n",
01054                     i, optimize->thread_climbing[i]);
01055             #endif
01056         }
01057         for (i = 0; i < nthreads_climbing; ++i)
01058         {
01059             data[i].thread = i;
01060             thread[i] = g_thread_new
01061             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01062         }
01063         for (i = 0; i < nthreads_climbing; ++i)
01064             g_thread_join (thread[i]);
01065     }
01066     #if DEBUG_OPTIMIZE
01067     fprintf (stderr, "optimize_step_climbing: end\n");
01068     #endif
01069 }
01070
01071 static inline void
01072 optimize_climbing_best ()
01073 {
01074     #if DEBUG_OPTIMIZE
01075     fprintf (stderr, "optimize_climbing_best: start\n");
01076     #endif
01077     optimize->simulation_best[0] = 0;
01078     memcpy (optimize->value, optimize->value_old,
01079            optimize->nvariables * sizeof (double));
01080     #if DEBUG_OPTIMIZE
01081     fprintf (stderr, "optimize_climbing_best: end\n");
01082     #endif
01083 }
01084
01085 static inline void
01086 optimize_climbing (unsigned int nsteps)
01087 {
01088     unsigned int i, j, k, b, s, adjust;
01089     #if DEBUG_OPTIMIZE
01090     fprintf (stderr, "optimize_climbing: start\n");
01091     #endif
01092     for (i = 0; i < optimize->nvariables; ++i)
01093         optimize->climbing[i] = 0.;

```

```

01100     b = optimize->simulation_best[0] * optimize->nvariables;
01101     s = optimize->nsimulations;
01102     adjust = 1;
01103     for (i = 0; i < nsteps; ++i, s += optimize->nestimates, b = k)
01104     {
01105         #if DEBUG_OPTIMIZE
01106             fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01107                     i, optimize->simulation_best[0]);
01108         #endif
01109         optimize_step_climbing (s);
01110         k = optimize->simulation_best[0] * optimize->nvariables;
01111         #if DEBUG_OPTIMIZE
01112             fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01113                     i, optimize->simulation_best[0]);
01114         #endif
01115         if (k == b)
01116         {
01117             if (adjust)
01118                 for (j = 0; j < optimize->nvariables; ++j)
01119                     optimize->step[j] *= 0.5;
01120             for (j = 0; j < optimize->nvariables; ++j)
01121                 optimize->climbing[j] = 0.;
01122             adjust = 1;
01123         }
01124         else
01125         {
01126             for (j = 0; j < optimize->nvariables; ++j)
01127             {
01128                 #if DEBUG_OPTIMIZE
01129                     fprintf (stderr,
01130                             "optimize_climbing: best%u=%%.14le old%u=%%.14le\n",
01131                             j, optimize->value[k + j], j, optimize->value[b + j]);
01132                 #endif
01133                 optimize->climbing[j]
01134                     = (1. - optimize->relaxation) * optimize->climbing[j]
01135                     + optimize->relaxation
01136                     * (optimize->value[k + j] - optimize->value[b + j]);
01137                 #if DEBUG_OPTIMIZE
01138                     fprintf (stderr, "optimize_climbing: climbing%u=%%.14le\n",
01139                             j, optimize->climbing[j]);
01140                 #endif
01141             }
01142             adjust = 0;
01143         }
01144     }
01145     #if DEBUG_OPTIMIZE
01146         fprintf (stderr, "optimize_climbing: end\n");
01147     #endif
01148 }
01149
01150 static double
01151 optimize_genetic_objective (Entity * entity)
01152 {
01153     unsigned int j;
01154     double objective;
01155     char buffer[64];
01156     #if DEBUG_OPTIMIZE
01157         fprintf (stderr, "optimize_genetic_objective: start\n");
01158     #endif
01159     for (j = 0; j < optimize->nvariables; ++j)
01160     {
01161         optimize->value[entity->id * optimize->nvariables + j]
01162             = genetic_get_variable (entity, optimize->genetic_variable + j);
01163     }
01164     objective = optimize_norm (entity->id);
01165     g_mutex_lock (mutex);
01166     for (j = 0; j < optimize->nvariables; ++j)
01167     {
01168         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01169         fprintf (optimize->file_variables, buffer,
01170                 genetic_get_variable (entity, optimize->genetic_variable + j));
01171     }
01172     fprintf (optimize->file_variables, "%.14le\n", objective);
01173     g_mutex_unlock (mutex);
01174     #if DEBUG_OPTIMIZE
01175         fprintf (stderr, "optimize_genetic_objective: end\n");
01176     #endif
01177     return objective;
01178 }
01179
01180 static void
01181 optimize_genetic ()
01182 {
01183     double *best_variable = NULL;
01184     char *best_genome = NULL;
01185     double best_objective = 0.;
01186     #if DEBUG_OPTIMIZE

```



```

01195     fprintf (stderr, "optimize_genetic: start\n");
01196     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01197             nthreads);
01198     fprintf (stderr,
01199             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01200             optimize->nvariables, optimize->nsimulations, optimize->niterations);
01201     fprintf (stderr,
01202             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01203             optimize->mutation_ratio, optimize->reproduction_ratio,
01204             optimize->adaptation_ratio);
01205 #endif
01206     genetic_algorithm_default (optimize->nvariables,
01207                               optimize->genetic_variable,
01208                               optimize->nsimulations,
01209                               optimize->niterations,
01210                               optimize->mutation_ratio,
01211                               optimize->reproduction_ratio,
01212                               optimize->adaptation_ratio,
01213                               optimize->seed,
01214                               optimize->threshold,
01215                               &optimize_genetic_objective,
01216                               &best_genome, &best_variable, &best_objective);
01217 #if DEBUG_OPTIMIZE
01218     fprintf (stderr, "optimize_genetic: the best\n");
01219 #endif
01220     optimize->error_old = (double *) g_malloc (sizeof (double));
01221     optimize->value_old
01222     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01223     optimize->error_old[0] = best_objective;
01224     memcpy (optimize->value_old, best_variable,
01225            optimize->nvariables * sizeof (double));
01226     g_free (best_genome);
01227     g_free (best_variable);
01228     optimize_print ();
01229 #if DEBUG_OPTIMIZE
01230     fprintf (stderr, "optimize_genetic: end\n");
01231 #endif
01232 }
01233
01237 static inline void
01238 optimize_save_old ()
01239 {
01240     unsigned int i, j;
01241 #if DEBUG_OPTIMIZE
01242     fprintf (stderr, "optimize_save_old: start\n");
01243     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01244 #endif
01245     memcpy (optimize->error_old, optimize->error_best,
01246            optimize->nbest * sizeof (double));
01247     for (i = 0; i < optimize->nbest; ++i)
01248     {
01249         j = optimize->simulation_best[i];
01250 #if DEBUG_OPTIMIZE
01251         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01252 #endif
01253         memcpy (optimize->value_old + i * optimize->nvariables,
01254                optimize->value + j * optimize->nvariables,
01255                optimize->nvariables * sizeof (double));
01256     }
01257 #if DEBUG_OPTIMIZE
01258     for (i = 0; i < optimize->nvariables; ++i)
01259         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01260                 i, optimize->value_old[i]);
01261     fprintf (stderr, "optimize_save_old: end\n");
01262 #endif
01263 }
01264
01269 static inline void
01270 optimize_merge_old ()
01271 {
01272     unsigned int i, j, k;
01273     double v[optimize->nbest * optimize->nvariables], e[optimize->nbest],
01274            *enew, *eold;
01275 #if DEBUG_OPTIMIZE
01276     fprintf (stderr, "optimize_merge_old: start\n");
01277 #endif
01278     anew = optimize->error_best;
01279     eold = optimize->error_old;
01280     i = j = k = 0;
01281     do
01282     {
01283         if (*enew < *eold)
01284         {
01285             memcpy (v + k * optimize->nvariables,
01286                    optimize->value
01287                    + optimize->simulation_best[i] * optimize->nvariables,
01288                    optimize->nvariables * sizeof (double));

```

```

01289         e[k] = *enew;
01290         ++k;
01291         ++enew;
01292         ++i;
01293     }
01294     else
01295     {
01296         memcpy (v + k * optimize->nvariables,
01297                 optimize->value_old + j * optimize->nvariables,
01298                 optimize->nvariables * sizeof (double));
01299         e[k] = *eold;
01300         ++k;
01301         ++eold;
01302         ++j;
01303     }
01304 }
01305 while (k < optimize->nbest);
01306 memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01307 memcpy (optimize->error_old, e, k * sizeof (double));
01308 #if DEBUG_OPTIMIZE
01309     fprintf (stderr, "optimize_merge_old: end\n");
01310 #endif
01311 }
01312
01313 static inline void
01314 optimize_refine ()
01315 {
01316     unsigned int i, j;
01317     double d;
01318     #if HAVE_MPI
01319     MPI_Status mpi_stat;
01320     #endif
01321     #if DEBUG_OPTIMIZE
01322     fprintf (stderr, "optimize_refine: start\n");
01323     #endif
01324     #if HAVE_MPI
01325     if (!optimize->mpi_rank)
01326     {
01327         #endif
01328         for (j = 0; j < optimize->nvariables; ++j)
01329         {
01330             optimize->rangemin[j] = optimize->rangemax[j]
01331             = optimize->value_old[j];
01332             optimize->step[j] = input->variable[j].step;
01333         }
01334         for (i = 0; ++i < optimize->nbest;)
01335         {
01336             for (j = 0; j < optimize->nvariables; ++j)
01337             {
01338                 optimize->rangemin[j]
01339                 = fmin (optimize->rangemin[j],
01340                         optimize->value_old[i * optimize->nvariables + j]);
01341                 optimize->rangemax[j]
01342                 = fmax (optimize->rangemax[j],
01343                         optimize->value_old[i * optimize->nvariables + j]);
01344             }
01345             for (j = 0; j < optimize->nvariables; ++j)
01346             {
01347                 d = optimize->tolerance
01348                 * (optimize->rangemax[j] - optimize->rangemin[j]);
01349                 switch (optimize->algorithm)
01350                 {
01351                     case ALGORITHM_MONTE_CARLO:
01352                         d *= 0.5;
01353                         break;
01354                     default:
01355                         if (optimize->nsweeps[j] > 1)
01356                             d /= optimize->nsweeps[j] - 1;
01357                         else
01358                             d = 0.;
01359                 }
01360                 optimize->rangemin[j] -= d;
01361                 optimize->rangemin[j]
01362                 = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01363                 optimize->rangemax[j] += d;
01364                 optimize->rangemax[j]
01365                 = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01366                 printf ("%s min=%lg max=%lg\n", optimize->label[j],
01367                         optimize->rangemin[j], optimize->rangemax[j]);
01368                 fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01369                         optimize->label[j], optimize->rangemin[j],
01370                         optimize->rangemax[j]);
01371             }
01372         }
01373     }
01374     #if HAVE_MPI
01375     for (i = 1; (int) i < ntasks; ++i)
01376     {

```

```

01380         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01381                   1, MPI_COMM_WORLD);
01382         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01383                   1, MPI_COMM_WORLD);
01384     }
01385 }
01386 else
01387 {
01388     MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01389              MPI_COMM_WORLD, &mpi_stat);
01390     MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01391              MPI_COMM_WORLD, &mpi_stat);
01392 }
01393 #endif
01394 #if DEBUG_OPTIMIZE
01395 fprintf (stderr, "optimize_refine: end\n");
01396 #endif
01397 }
01398
01402 static void
01403 optimize_step ()
01404 {
01405     #if DEBUG_OPTIMIZE
01406     fprintf (stderr, "optimize_step: start\n");
01407     #endif
01408     optimize_algorithm ();
01409     if (optimize->nsteps)
01410         optimize_climbing (optimize->nsteps);
01411     #if DEBUG_OPTIMIZE
01412     fprintf (stderr, "optimize_step: end\n");
01413     #endif
01414 }
01415
01419 static inline void
01420 optimize_iterate ()
01421 {
01422     unsigned int i;
01423     #if DEBUG_OPTIMIZE
01424     fprintf (stderr, "optimize_iterate: start\n");
01425     #endif
01426     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01427     optimize->value_old =
01428         (double *) g_malloc (optimize->nbest * optimize->nvariables *
01429                             sizeof (double));
01430     optimize_step ();
01431     optimize_save_old ();
01432     optimize_refine ();
01433     optimize_print ();
01434     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01435     {
01436         optimize_step ();
01437         optimize_merge_old ();
01438         optimize_refine ();
01439         optimize_print ();
01440     }
01441     if (optimize->nfinal_steps && !optimize->stop)
01442     {
01443         optimize_climbing_best ();
01444         optimize_climbing (optimize->nfinal_steps);
01445         optimize_merge_old ();
01446         optimize_print ();
01447     }
01448     #if DEBUG_OPTIMIZE
01449     fprintf (stderr, "optimize_iterate: end\n");
01450     #endif
01451 }
01452
01456 static inline void
01457 optimize_save_optimal ()
01458 {
01459     char cinput[32];
01460     unsigned int i, j;
01461     unsigned int flags = 1;
01462
01463     // Getting optimal values
01464     memcpy (optimize->value, optimize->value_old,
01465            optimize->nvariables * sizeof (double));
01466
01467     // Saving optimal input files
01468     for (i = 0; i < optimize->ninputs; ++i)
01469         for (j = 0; j < optimize->nexperiments; ++j)
01470         {
01471             snprintf (cinput, 32, "optimal-%u-%u", i, j);
01472             #if DEBUG_OPTIMIZE
01473             fprintf (stderr, "optimize_save_optimal: i=%u j=%u input=%s\n",
01474                     i, j, cinput);
01475             #endif

```

```

01476         // Checking templates
01477         if (optimize->template_flags & flags)
01478             optimize_input (0, cinput, optimize->file[i][j]);
01479         flags <= 1;
01480     }
01481 }
01482
01486 void
01487 optimize_free ()
01488 {
01489     unsigned int i, j;
01490     #if DEBUG_OPTIMIZE
01491     fprintf (stderr, "optimize_free: start\n");
01492     #endif
01493     for (j = 0; j < optimize->ninputs; ++j)
01494     {
01495         for (i = 0; i < optimize->nexperiments; ++i)
01496             g_mapped_file_unref (optimize->file[j][i]);
01497         g_free (optimize->file[j]);
01498     }
01499     g_free (optimize->error_old);
01500     g_free (optimize->value_old);
01501     g_free (optimize->value);
01502     g_free (optimize->genetic_variable);
01503     #if DEBUG_OPTIMIZE
01504     fprintf (stderr, "optimize_free: end\n");
01505     #endif
01506 }
01507
01511 void
01512 optimize_open ()
01513 {
01514     GTimeZone *tz;
01515     GDateTime *t0, *t;
01516     unsigned int i, j, nsteps;
01517
01518     #if DEBUG_OPTIMIZE
01519     char *buffer;
01520     fprintf (stderr, "optimize_open: start\n");
01521     #endif
01522
01523     // Getting initial time
01524     #if DEBUG_OPTIMIZE
01525     fprintf (stderr, "optimize_open: getting initial time\n");
01526     #endif
01527     tz = g_time_zone_new_utc ();
01528     t0 = g_date_time_new_now (tz);
01529
01530     // Obtaining and initing the pseudo-random numbers generator seed
01531     #if DEBUG_OPTIMIZE
01532     fprintf (stderr, "optimize_open: getting initial seed\n");
01533     #endif
01534     if (optimize->seed == DEFAULT_RANDOM_SEED)
01535         optimize->seed = input->seed;
01536     gsl_rng_set (optimize->rng, optimize->seed);
01537
01538     // Obtaining template flags
01539     #if DEBUG_OPTIMIZE
01540     fprintf (stderr, "optimize_open: getting template flags\n");
01541     #endif
01542     optimize->template_flags = input->template_flags;
01543
01544     // Replacing the working directory
01545     #if DEBUG_OPTIMIZE
01546     fprintf (stderr, "optimize_open: replacing the working directory\n");
01547     #endif
01548     g_chdir (input->directory);
01549
01550     // Getting results file names
01551     optimize->result = input->result;
01552     optimize->variables = input->variables;
01553
01554     // Obtaining the simulator file
01555     optimize->simulator = input->simulator;
01556
01557     // Obtaining the evaluator file
01558     optimize->evaluator = input->evaluator;
01559
01560     // Obtaining the cleaner file
01561     optimize->cleaner = input->cleaner;
01562
01563     // Reading the algorithm
01564     optimize->algorithm = input->algorithm;
01565     switch (optimize->algorithm)
01566     {
01567     case ALGORITHM_MONTE_CARLO:
01568         optimize_algorithm = optimize_MonteCarlo;

```

```

01569         break;
01570     case ALGORITHM_SWEEP:
01571         optimize_algorithm = optimize_sweep;
01572         break;
01573     case ALGORITHM_ORTHOGONAL:
01574         optimize_algorithm = optimize_orthogonal;
01575         break;
01576     default:
01577         optimize_algorithm = optimize_genetic;
01578         optimize->mutation_ratio = input->mutation_ratio;
01579         optimize->reproduction_ratio = input->reproduction_ratio;
01580         optimize->adaptation_ratio = input->adaptation_ratio;
01581     }
01582     optimize->nvariables = input->nvariables;
01583     optimize->nsimulations = input->nsimulations;
01584     optimize->niterations = input->niterations;
01585     optimize->nbest = input->nbest;
01586     optimize->tolerance = input->tolerance;
01587     optimize->nsteps = input->nsteps;
01588     optimize->nfinal_steps = input->nfinal_steps;
01589     nsteps = JBM_MAX (optimize->nsteps, optimize->nfinal_steps);
01590     optimize->nestimates = 0;
01591     optimize->threshold = input->threshold;
01592     optimize->stop = 0;
01593     if (nsteps)
01594     {
01595         optimize->relaxation = input->relaxation;
01596         switch (input->climbing)
01597         {
01598             case CLIMBING_METHOD_COORDINATES:
01599                 optimize->nestimates = 2 * optimize->nvariables;
01600                 optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
01601                 break;
01602             default:
01603                 optimize->nestimates = input->nestimates;
01604                 optimize_estimate_climbing = optimize_estimate_climbing_random;
01605         }
01606     }
01607
01608 #if DEBUG_OPTIMIZE
01609     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01610 #endif
01611     optimize->simulation_best
01612     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01613     optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01614
01615     // Reading the experimental data
01616 #if DEBUG_OPTIMIZE
01617     buffer = g_get_current_dir ();
01618     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01619     g_free (buffer);
01620 #endif
01621     optimize->nexperiments = input->nexperiments;
01622     optimize->ninputs = input->experiment->ninputs;
01623     optimize->experiment
01624     = (char **) alloca (input->nexperiments * sizeof (char *));
01625     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01626     for (i = 0; i < input->experiment->ninputs; ++i)
01627         optimize->file[i] = (GMappedFile **)
01628             g_malloc (input->nexperiments * sizeof (GMappedFile *));
01629     for (i = 0; i < input->nexperiments; ++i)
01630     {
01631         #if DEBUG_OPTIMIZE
01632             fprintf (stderr, "optimize_open: i=%u\n", i);
01633         #endif
01634         optimize->experiment[i] = input->experiment[i].name;
01635         optimize->weight[i] = input->experiment[i].weight;
01636         #if DEBUG_OPTIMIZE
01637             fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01638                     optimize->experiment[i], optimize->weight[i]);
01639         #endif
01640         for (j = 0; j < input->experiment->ninputs; ++j)
01641         {
01642             #if DEBUG_OPTIMIZE
01643                 fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01644             #endif
01645             optimize->file[j][i]
01646             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01647         }
01648     }
01649
01650     // Reading the variables data
01651 #if DEBUG_OPTIMIZE
01652     fprintf (stderr, "optimize_open: reading variables\n");
01653 #endif
01654     optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01655     j = input->nvariables * sizeof (double);

```

```

01656 optimize->rangemin = (double *) alloca (j);
01657 optimize->rangeminabs = (double *) alloca (j);
01658 optimize->rangemax = (double *) alloca (j);
01659 optimize->rangemaxabs = (double *) alloca (j);
01660 optimize->step = (double *) alloca (j);
01661 j = input->nvariables * sizeof (unsigned int);
01662 optimize->precision = (unsigned int *) alloca (j);
01663 optimize->nsweeps = (unsigned int *) alloca (j);
01664 optimize->nbits = (unsigned int *) alloca (j);
01665 for (i = 0; i < input->nvariables; ++i)
01666 {
01667     optimize->label[i] = input->variable[i].name;
01668     optimize->rangemin[i] = input->variable[i].rangemin;
01669     optimize->rangeminabs[i] = input->variable[i].rangeminabs;
01670     optimize->rangemax[i] = input->variable[i].rangemax;
01671     optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
01672     optimize->precision[i] = input->variable[i].precision;
01673     optimize->step[i] = input->variable[i].step;
01674     optimize->nsweeps[i] = input->variable[i].nsweeps;
01675     optimize->nbits[i] = input->variable[i].nbits;
01676 }
01677 if (input->algorithm == ALGORITHM_SWEEP
01678     || input->algorithm == ALGORITHM_ORTHOGONAL)
01679 {
01680     optimize->nsimulations = 1;
01681     for (i = 0; i < input->nvariables; ++i)
01682     {
01683         optimize->nsimulations *= optimize->nsweeps[i];
01684 #if DEBUG_OPTIMIZE
01685         fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01686                 optimize->nsweeps[i], optimize->nsimulations);
01687 #endif
01688     }
01689 }
01690 if (nsteps)
01691     optimize->climbing
01692     = (double *) alloca (optimize->nvariables * sizeof (double));
01693 // Setting error norm
01694 switch (input->norm)
01695 {
01696     case ERROR_NORM_EUCLIDIAN:
01697         optimize_norm = optimize_norm_euclidian;
01698         break;
01699     case ERROR_NORM_MAXIMUM:
01700         optimize_norm = optimize_norm_maximum;
01701         break;
01702     case ERROR_NORM_P:
01703         optimize_norm = optimize_norm_p;
01704         optimize->p = input->p;
01705         break;
01706     default:
01707         optimize_norm = optimize_norm_taxicab;
01708 }
01709 // Allocating values
01710 #if DEBUG_OPTIMIZE
01711 fprintf (stderr, "optimize_open: allocating variables\n");
01712 fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01713         optimize->nvariables, optimize->algorithm);
01714 #endif
01715 optimize->genetic_variable = NULL;
01716 if (optimize->algorithm == ALGORITHM_GENETIC)
01717 {
01718     optimize->genetic_variable = (GeneticVariable *)
01719     g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01720     for (i = 0; i < optimize->nvariables; ++i)
01721     {
01722         #if DEBUG_OPTIMIZE
01723             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01724                     i, optimize->rangemin[i], optimize->rangemax[i],
01725                     optimize->nbits[i]);
01726         #endif
01727         optimize->genetic_variable[i].minimum = optimize->rangemin[i];
01728         optimize->genetic_variable[i].maximum = optimize->rangemax[i];
01729         optimize->genetic_variable[i].nbits = optimize->nbits[i];
01730     }
01731 }
01732 #if DEBUG_OPTIMIZE
01733 fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01734         optimize->nvariables, optimize->nsimulations);
01735 #endif
01736 optimize->value = (double *)
01737 g_malloc ((optimize->nsimulations + optimize->nestimates * nsteps)
01738         * optimize->nvariables * sizeof (double));
01739 // Calculating simulations to perform for each task

```

```

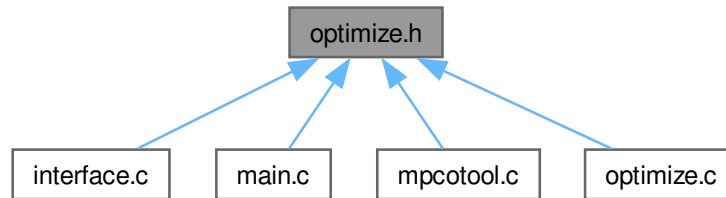
01743 #if HAVE_MPI
01744 #if DEBUG_OPTIMIZE
01745     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01746             optimize->mpi_rank, ntasks);
01747 #endif
01748     optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
01749     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
01750     if (nsteps)
01751     {
01752         optimize->nstart_climbing
01753             = optimize->mpi_rank * optimize->nestimates / ntasks;
01754         optimize->nend_climbing
01755             = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
01756     }
01757 #else
01758     optimize->nstart = 0;
01759     optimize->nend = optimize->nsimulations;
01760     if (nsteps)
01761     {
01762         optimize->nstart_climbing = 0;
01763         optimize->nend_climbing = optimize->nestimates;
01764     }
01765 #endif
01766 #if DEBUG_OPTIMIZE
01767     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01768             optimize->nend);
01769 #endif
01770
01771     // Calculating simulations to perform for each thread
01772     optimize->thread
01773         = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01774     for (i = 0; i <= nthreads; ++i)
01775     {
01776         optimize->thread[i] = optimize->nstart
01777             + i * (optimize->nend - optimize->nstart) / nthreads;
01778 #if DEBUG_OPTIMIZE
01779         fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01780                 optimize->thread[i]);
01781 #endif
01782     }
01783     if (nsteps)
01784         optimize->thread_climbing = (unsigned int *)
01785             alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01786
01787     // Opening result files
01788     optimize->file_result = g_fopen (optimize->result, "w");
01789     optimize->file_variables = g_fopen (optimize->variables, "w");
01790
01791     // Performing the algorithm
01792     switch (optimize->algorithm)
01793     {
01794         // Genetic algorithm
01795         case ALGORITHM_GENETIC:
01796             optimize_genetic ();
01797             break;
01798
01799         // Iterative algorithm
01800         default:
01801             optimize_iterate ();
01802     }
01803
01804     // Getting calculation time
01805     t = g_date_time_new_now (tz);
01806     optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01807     g_date_time_unref (t);
01808     g_date_time_unref (t0);
01809     g_time_zone_unref (tz);
01810     printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01811     fprintf (optimize->file_result, "%s = %.6lg s\n",
01812             _("Calculation time"), optimize->calculation_time);
01813
01814     // Closing result files
01815     optimize_save_optimal ();
01816     fclose (optimize->file_variables);
01817     fclose (optimize->file_result);
01818
01819 #if DEBUG_OPTIMIZE
01820     fprintf (stderr, "optimize_open: end\n");
01821 #endif
01822 }

```

## 4.23 optimize.h File Reference

Header file to define the optimization functions.

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



### Data Structures

- struct [Optimize](#)  
*Struct to define the optimization ation data.*
- struct [ParallelData](#)  
*Struct to pass to the GThreads parallelized function.*

### Functions

- void [optimize\\_free](#) ()
- void [optimize\\_open](#) ()

### Variables

- int **ntasks**
- unsigned int **nthreads**
- unsigned int [nthreads\\_climbing](#)  
*Number of threads for the hill climbing method.*
- GMutex **mutex** [1]
- [Optimize](#) **optimize** [1]  
*Optimization data.*

### 4.23.1 Detailed Description

Header file to define the optimization functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [optimize.h](#).



## 4.23.2 Function Documentation

### 4.23.2.1 optimize\_free()

```
void optimize_free ( )
```

Function to free the memory used by the [Optimize](#) struct.

Definition at line 1487 of file [optimize.c](#).

```
01488 {
01489     unsigned int i, j;
01490     #if DEBUG_OPTIMIZE
01491     fprintf (stderr, "optimize_free: start\n");
01492     #endif
01493     for (j = 0; j < optimize->ninputs; ++j)
01494     {
01495         for (i = 0; i < optimize->nexperiments; ++i)
01496             g_mapped_file_unref (optimize->file[j][i]);
01497         g_free (optimize->file[j]);
01498     }
01499     g_free (optimize->error_old);
01500     g_free (optimize->value_old);
01501     g_free (optimize->value);
01502     g_free (optimize->genetic_variable);
01503     #if DEBUG_OPTIMIZE
01504     fprintf (stderr, "optimize_free: end\n");
01505     #endif
01506 }
```

### 4.23.2.2 optimize\_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1512 of file [optimize.c](#).

```
01513 {
01514     GTimeZone *tz;
01515     GDateTime *t0, *t;
01516     unsigned int i, j, nsteps;
01517     #if DEBUG_OPTIMIZE
01518     char *buffer;
01519     fprintf (stderr, "optimize_open: start\n");
01520     #endif
01521     // Getting initial time
01522     #if DEBUG_OPTIMIZE
01523     fprintf (stderr, "optimize_open: getting initial time\n");
01524     #endif
01525     tz = g_time_zone_new_utc ();
01526     t0 = g_date_time_new_now (tz);
01527     // Obtaining and initing the pseudo-random numbers generator seed
01528     #if DEBUG_OPTIMIZE
01529     fprintf (stderr, "optimize_open: getting initial seed\n");
01530     #endif
01531     if (optimize->seed == DEFAULT_RANDOM_SEED)
01532         optimize->seed = input->seed;
01533     gsl_rng_set (optimize->rng, optimize->seed);
01534     // Obtaining template flags
01535     #if DEBUG_OPTIMIZE
01536     fprintf (stderr, "optimize_open: getting template flags\n");
01537     #endif
01538     optimize->template_flags = input->template_flags;
01539     // Replacing the working directory
01540     #if DEBUG_OPTIMIZE
01541     fprintf (stderr, "optimize_open: replacing the working directory\n");
01542     #endif
01543     g_chdir (input->directory);
01544     // Getting results file names
01545     optimize->result = input->result;
```

```

01552     optimize->variables = input->variables;
01553
01554     // Obtaining the simulator file
01555     optimize->simulator = input->simulator;
01556
01557     // Obtaining the evaluator file
01558     optimize->evaluator = input->evaluator;
01559
01560     // Obtaining the cleaner file
01561     optimize->cleaner = input->cleaner;
01562
01563     // Reading the algorithm
01564     optimize->algorithm = input->algorithm;
01565     switch (optimize->algorithm)
01566     {
01567         case ALGORITHM_MONTE_CARLO:
01568             optimize_algorithm = optimize_MonteCarlo;
01569             break;
01570         case ALGORITHM_SWEEP:
01571             optimize_algorithm = optimize_sweep;
01572             break;
01573         case ALGORITHM_ORTHOGONAL:
01574             optimize_algorithm = optimize_orthogonal;
01575             break;
01576         default:
01577             optimize_algorithm = optimize_genetic;
01578             optimize->mutation_ratio = input->mutation_ratio;
01579             optimize->reproduction_ratio = input->reproduction_ratio;
01580             optimize->adaptation_ratio = input->adaptation_ratio;
01581     }
01582     optimize->nvariables = input->nvariables;
01583     optimize->nsimulations = input->nsimulations;
01584     optimize->niterations = input->niterations;
01585     optimize->nbest = input->nbest;
01586     optimize->tolerance = input->tolerance;
01587     optimize->nsteps = input->nsteps;
01588     optimize->nfinal_steps = input->nfinal_steps;
01589     nsteps = JBM_MAX (optimize->nsteps, optimize->nfinal_steps);
01590     optimize->nestimates = 0;
01591     optimize->threshold = input->threshold;
01592     optimize->stop = 0;
01593     if (nsteps)
01594     {
01595         optimize->relaxation = input->relaxation;
01596         switch (input->climbing)
01597         {
01598             case CLIMBING_METHOD_COORDINATES:
01599                 optimize->nestimates = 2 * optimize->nvariables;
01600                 optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
01601                 break;
01602             default:
01603                 optimize->nestimates = input->nestimates;
01604                 optimize_estimate_climbing = optimize_estimate_climbing_random;
01605         }
01606     }
01607
01608 #if DEBUG_OPTIMIZE
01609     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01610 #endif
01611     optimize->simulation_best
01612     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01613     optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01614
01615     // Reading the experimental data
01616 #if DEBUG_OPTIMIZE
01617     buffer = g_get_current_dir ();
01618     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01619     g_free (buffer);
01620 #endif
01621     optimize->nexperiments = input->nexperiments;
01622     optimize->ninputs = input->experiment->ninputs;
01623     optimize->experiment
01624     = (char **) alloca (input->nexperiments * sizeof (char *));
01625     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01626     for (i = 0; i < input->experiment->ninputs; ++i)
01627         optimize->file[i] = (GMappedFile **)
01628             g_malloc (input->nexperiments * sizeof (GMappedFile *));
01629     for (i = 0; i < input->nexperiments; ++i)
01630     {
01631 #if DEBUG_OPTIMIZE
01632         fprintf (stderr, "optimize_open: i=%u\n", i);
01633 #endif
01634         optimize->experiment[i] = input->experiment[i].name;
01635         optimize->weight[i] = input->experiment[i].weight;
01636 #if DEBUG_OPTIMIZE
01637         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01638             optimize->experiment[i], optimize->weight[i]);

```

```

01639 #endif
01640     for (j = 0; j < input->experiment->ninputs; ++j)
01641     {
01642 #if DEBUG_OPTIMIZE
01643         fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01644 #endif
01645         optimize->file[j][i]
01646             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01647     }
01648 }
01649
01650 // Reading the variables data
01651 #if DEBUG_OPTIMIZE
01652 fprintf (stderr, "optimize_open: reading variables\n");
01653 #endif
01654 optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01655 j = input->nvariables * sizeof (double);
01656 optimize->rangemin = (double *) alloca (j);
01657 optimize->rangeminabs = (double *) alloca (j);
01658 optimize->rangemax = (double *) alloca (j);
01659 optimize->rangemaxabs = (double *) alloca (j);
01660 optimize->step = (double *) alloca (j);
01661 j = input->nvariables * sizeof (unsigned int);
01662 optimize->precision = (unsigned int *) alloca (j);
01663 optimize->nsweeps = (unsigned int *) alloca (j);
01664 optimize->nbits = (unsigned int *) alloca (j);
01665 for (i = 0; i < input->nvariables; ++i)
01666 {
01667     optimize->label[i] = input->variable[i].name;
01668     optimize->rangemin[i] = input->variable[i].rangemin;
01669     optimize->rangeminabs[i] = input->variable[i].rangeminabs;
01670     optimize->rangemax[i] = input->variable[i].rangemax;
01671     optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
01672     optimize->precision[i] = input->variable[i].precision;
01673     optimize->step[i] = input->variable[i].step;
01674     optimize->nsweeps[i] = input->variable[i].nsweeps;
01675     optimize->nbits[i] = input->variable[i].nbits;
01676 }
01677 if (input->algorithm == ALGORITHM_SWEEP
01678     || input->algorithm == ALGORITHM_ORTHOGONAL)
01679 {
01680     optimize->nsimulations = 1;
01681     for (i = 0; i < input->nvariables; ++i)
01682     {
01683         optimize->nsimulations *= optimize->nsweeps[i];
01684 #if DEBUG_OPTIMIZE
01685         fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01686             optimize->nsweeps[i], optimize->nsimulations);
01687 #endif
01688     }
01689 }
01690 if (nsteps)
01691     optimize->climbing
01692         = (double *) alloca (optimize->nvariables * sizeof (double));
01693
01694 // Setting error norm
01695 switch (input->norm)
01696 {
01697     case ERROR_NORM_EUCLIDIAN:
01698         optimize_norm = optimize_norm_euclidian;
01699         break;
01700     case ERROR_NORM_MAXIMUM:
01701         optimize_norm = optimize_norm_maximum;
01702         break;
01703     case ERROR_NORM_P:
01704         optimize_norm = optimize_norm_p;
01705         optimize->p = input->p;
01706         break;
01707     default:
01708         optimize_norm = optimize_norm_taxicab;
01709 }
01710
01711 // Allocating values
01712 #if DEBUG_OPTIMIZE
01713 fprintf (stderr, "optimize_open: allocating variables\n");
01714 fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01715     optimize->nvariables, optimize->algorithm);
01716 #endif
01717 optimize->genetic_variable = NULL;
01718 if (optimize->algorithm == ALGORITHM_GENETIC)
01719 {
01720     optimize->genetic_variable = (GeneticVariable *)
01721         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01722     for (i = 0; i < optimize->nvariables; ++i)
01723     {
01724 #if DEBUG_OPTIMIZE
01725         fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",

```

```

01726         i, optimize->rangemin[i], optimize->rangemax[i],
01727         optimize->nbits[i]);
01728 #endif
01729     optimize->genetic_variable[i].minimum = optimize->rangemin[i];
01730     optimize->genetic_variable[i].maximum = optimize->rangemax[i];
01731     optimize->genetic_variable[i].nbits = optimize->nbits[i];
01732 }
01733 }
01734 #if DEBUG_OPTIMIZE
01735 fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01736         optimize->nvariables, optimize->nsimulations);
01737 #endif
01738 optimize->value = (double *)
01739     g_malloc ((optimize->nsimulations + optimize->nestimates * nsteps)
01740             * optimize->nvariables * sizeof (double));
01741
01742 // Calculating simulations to perform for each task
01743 #if HAVE_MPI
01744 #if DEBUG_OPTIMIZE
01745 fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01746         optimize->mpi_rank, ntasks);
01747 #endif
01748 optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
01749 optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
01750 if (nsteps)
01751 {
01752     optimize->nstart_climbing
01753         = optimize->mpi_rank * optimize->nestimates / ntasks;
01754     optimize->nend_climbing
01755         = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
01756 }
01757 #else
01758 optimize->nstart = 0;
01759 optimize->nend = optimize->nsimulations;
01760 if (nsteps)
01761 {
01762     optimize->nstart_climbing = 0;
01763     optimize->nend_climbing = optimize->nestimates;
01764 }
01765 #endif
01766 #if DEBUG_OPTIMIZE
01767 fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01768         optimize->nend);
01769 #endif
01770
01771 // Calculating simulations to perform for each thread
01772 optimize->thread
01773     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01774 for (i = 0; i <= nthreads; ++i)
01775 {
01776     optimize->thread[i] = optimize->nstart
01777         + i * (optimize->nend - optimize->nstart) / nthreads;
01778 #if DEBUG_OPTIMIZE
01779 fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01780         optimize->thread[i]);
01781 #endif
01782 }
01783 if (nsteps)
01784     optimize->thread_climbing = (unsigned int *)
01785         alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01786
01787 // Opening result files
01788 optimize->file_result = g_fopen (optimize->result, "w");
01789 optimize->file_variables = g_fopen (optimize->variables, "w");
01790
01791 // Performing the algorithm
01792 switch (optimize->algorithm)
01793 {
01794     // Genetic algorithm
01795     case ALGORITHM_GENETIC:
01796         optimize_genetic ();
01797         break;
01798
01799     // Iterative algorithm
01800     default:
01801         optimize_iterate ();
01802 }
01803
01804 // Getting calculation time
01805 t = g_date_time_new_now (tz);
01806 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01807 g_date_time_unref (t);
01808 g_date_time_unref (t0);
01809 g_time_zone_unref (tz);
01810 printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01811 fprintf (optimize->file_result, "%s = %.6lg s\n",
01812         _("Calculation time"), optimize->calculation_time);

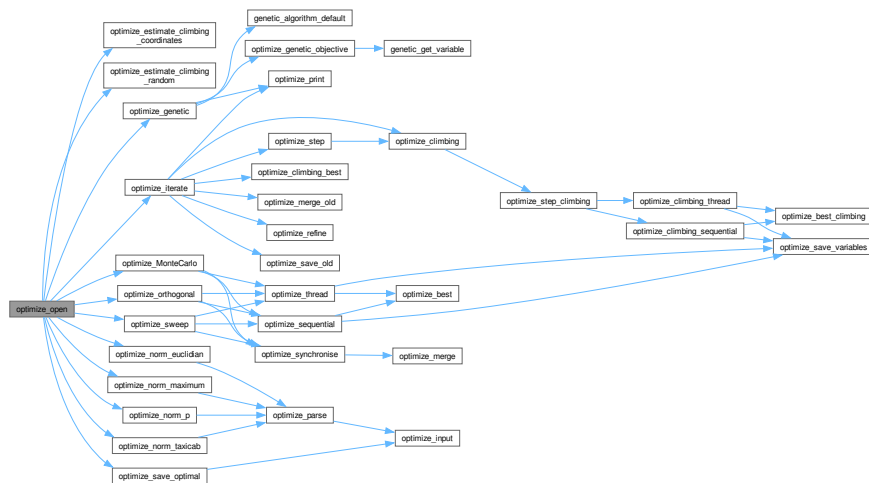
```

```

01813
01814 // Closing result files
01815 optimize_save_optimal ();
01816 fclose (optimize->file_variables);
01817 fclose (optimize->file_result);
01818
01819 #if DEBUG_OPTIMIZE
01820 fprintf (stderr, "optimize_open: end\n");
01821 #endif
01822 }

```

Here is the call graph for this function:



## 4.23.3 Variable Documentation

### 4.23.3.1 nthreads\_climbing

```
unsigned int nthreads_climbing [extern]
```

Number of threads for the hill climbing method.

Definition at line 84 of file [optimize.c](#).

### 4.23.3.2 optimize

```
Optimize optimize[1] [extern]
```

Optimization data.

Definition at line 83 of file [optimize.c](#).

## 4.24 optimize.h

[Go to the documentation of this file.](#)

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014      this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017      this list of conditions and the following disclaimer in the
00018      documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032 #ifndef OPTIMIZE__H
00033 #define OPTIMIZE__H 1
00034
00035 typedef struct
00036 {
00037     GMappedFile **file[MAX_NINPUTS];
00038     char **experiment;
00039     char **label;
00040     gsl_rng *rng;
00041     GeneticVariable *genetic_variable;
00042     FILE *file_result;
00043     FILE *file_variables;
00044     char *result;
00045     char *variables;
00046     char *simulator;
00047     char *evaluator;
00048     char *cleaner;
00049     double *value;
00050     double *rangemin;
00051     double *rangemax;
00052     double *rangeminabs;
00053     double *rangemaxabs;
00054     double *error_best;
00055     double *weight;
00056     double *step;
00057     double *climbing;
00058     double *value_old;
00059     double *error_old;
00060     unsigned int *precision;
00061     unsigned int *nsweeps;
00062     unsigned int *nbits;
00063     unsigned int *thread;
00064     unsigned int *thread_climbing;
00065     unsigned int *simulation_best;
00066     double tolerance;
00067     double mutation_ratio;
00068     double reproduction_ratio;
00069     double adaptation_ratio;
00070     double relaxation;
00071     double calculation_time;
00072     double p;
00073     double threshold;
00074     unsigned long int seed;
00075     unsigned int nvariables;
00076     unsigned int nexperiments;
00077     unsigned int ninputs;
00078     unsigned int nsimulations;
00079     unsigned int nsteps;
00080     unsigned int nfinal_steps;
00081     unsigned int nestimates;
00082     unsigned int algorithm;

```

```

00105 unsigned int nstart;
00106 unsigned int nend;
00107 unsigned int nstart_climbing;
00109 unsigned int nend_climbing;
00111 unsigned int niterations;
00112 unsigned int nbest;
00113 unsigned int nsaveds;
00114 unsigned int stop;
00115 unsigned int template_flags;
00116 #if HAVE_MPI
00117 int mpi_rank;
00118 #endif
00119 } Optimize;
00120
00125 typedef struct
00126 {
00127 unsigned int thread;
00128 } ParallelData;
00129
00130 // Global variables
00131 extern int ntasks;
00132 extern unsigned int nthreads;
00133 extern unsigned int nthreads_climbing;
00134 extern GMutex mutex[1];
00135 extern Optimize optimize[1];
00136
00137 // Public functions
00138 void optimize_free ();
00139 void optimize_open ();
00140
00141 #endif

```

## 4.25 tools.c File Reference

Source file to define some useful functions.

```

#include "config.h"
#include <stdio.h>
#include <unistd.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include <gtk/gtk.h>
#include "jb/src/jb_win.h"
#include "tools.h"

```

Include dependency graph for tools.c:



### Variables

- GtkWidget \* [main\\_window](#)  
Main GtkWidget.
- char \* [error\\_message](#)  
Error message.
- void(\* [show\\_pending](#) )() = NULL  
Pointer to the function to show pending events.

### 4.25.1 Detailed Description

Source file to define some useful functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [tools.c](#).

### 4.25.2 Variable Documentation

#### 4.25.2.1 error\_message

```
char* error_message
```

Error message.

Definition at line [59](#) of file [tools.c](#).

#### 4.25.2.2 main\_window

```
GtkWindow* main_window
```

Main GtkWindow.

Definition at line [56](#) of file [tools.c](#).

#### 4.25.2.3 show\_pending

```
void(* show_pending) () ( ) = NULL
```

Pointer to the function to show pending events.

Definition at line [60](#) of file [tools.c](#).



## 4.26 tools.c

[Go to the documentation of this file.](#)

```

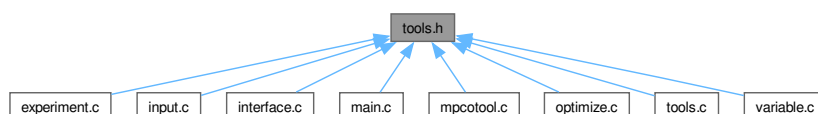
00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032  #define _GNU_SOURCE
00033  #include "config.h"
00034  #include <stdio.h>
00035  #include <unistd.h>
00036  #include <libxml/parser.h>
00037  #include <libintl.h>
00038  #include <glib.h>
00039  #include <json-glib/json-glib.h>
00040  #ifdef G_OS_WIN32
00041  #include <windows.h>
00042  #endif
00043  #if HAVE_GTK
00044  #include <gtk/gtk.h>
00045  #endif
00046  #include "jb/src/jb_win.h"
00047  #include "tools.h"
00048
00049  #if HAVE_GTK
00050  GtkWidget *main_window;
00051  #endif
00052  char *error_message;
00053  void (*show_pending) () = NULL;

```

## 4.27 tools.h File Reference

Header file to define some useful functions.

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



## Macros

- `#define ERROR_TYPE GTK_MESSAGE_ERROR`  
*Macro to define the error message type.*
- `#define INFO_TYPE GTK_MESSAGE_INFO`  
*Macro to define the information message type.*

## Variables

- `GtkWindow * main_window`  
*Main GtkWindow.*
- `GtkWindow * window_parent`
- `char * error_message`  
*Error message.*
- `void(* show_pending )()`  
*Pointer to the function to show pending events.*

### 4.27.1 Detailed Description

Header file to define some useful functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [tools.h](#).

### 4.27.2 Macro Definition Documentation

#### 4.27.2.1 ERROR\_TYPE

```
#define ERROR_TYPE GTK_MESSAGE_ERROR
```

Macro to define the error message type.

Definition at line [48](#) of file [tools.h](#).

#### 4.27.2.2 INFO\_TYPE

```
#define INFO_TYPE GTK_MESSAGE_INFO
```

Macro to define the information message type.

Definition at line [49](#) of file [tools.h](#).

### 4.27.3 Variable Documentation

#### 4.27.3.1 error\_message

```
char* error_message [extern]
```

Error message.

Definition at line 59 of file [tools.c](#).

#### 4.27.3.2 main\_window

```
GtkWindow* main_window [extern]
```

Main GtkWindow.

Definition at line 56 of file [tools.c](#).

#### 4.27.3.3 show\_pending

```
void(* show_pending) () ( ) [extern]
```

Pointer to the function to show pending events.

Definition at line 60 of file [tools.c](#).

## 4.28 tools.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef TOOLS__H
00033 #define TOOLS__H 1
00034
```

```

00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWidget *main_window;
00051 extern GtkWidget *window_parent;
00052 #else
00053 #define ERROR_TYPE 0
00054 #define INFO_TYPE 0
00055 #endif
00056
00057 // Public functions
00058
00059 extern char *error_message;
00060 extern void (*show_pending) ();
00061
00062 #endif

```

## 4.29 variable.c File Reference

Source file to define the variable data.

```

#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "jb/src/jb_xml.h"
#include "jb/src/jb_json.h"
#include "jb/src/jb_win.h"
#include "tools.h"
#include "variable.h"

```

Include dependency graph for variable.c:



### Macros

- `#define DEBUG_VARIABLE 0`  
Macro to debug variable functions.

### Functions

- void `variable_free` (Variable \*variable, unsigned int type)
- void `variable_error` (Variable \*variable, char \*message)
- int `variable_open_xml` (Variable \*variable, xmlNode \*node, unsigned int algorithm, unsigned int nsteps)
- int `variable_open_json` (Variable \*variable, JsonNode \*node, unsigned int algorithm, unsigned int nsteps)

### Variables

- const char \* `format` [NPRECISIONS]  
Array of C-strings with variable formats.
- const double `precision` [NPRECISIONS]  
Array of variable precisions.

## 4.29.1 Detailed Description

Source file to define the variable data.

### Authors

Javier Burguete and Borja Latorre.

### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [variable.c](#).

## 4.29.2 Macro Definition Documentation

### 4.29.2.1 DEBUG\_VARIABLE

```
#define DEBUG_VARIABLE 0
```

Macro to debug variable functions.

Definition at line 51 of file [variable.c](#).

## 4.29.3 Function Documentation

### 4.29.3.1 variable\_error()

```
void variable_error (  
    Variable * variable,  
    char * message )
```

Function to print a message error opening an [Variable](#) struct.

#### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>message</i>	Error message.

Definition at line 88 of file [variable.c](#).

```
00092 {  
00093     char buffer[64];  
00094     if (!variable->name)  
00095         snprintf (buffer, 64, "%s: %s", _("Variable"), message);  
00096     else  
00097         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);  
00098     error_message = g_strdup (buffer);  
00099 }
```

#### 4.29.3.2 variable\_free()

```
void variable_free (
    Variable * variable,
    unsigned int type )
```

Function to free the memory of a [Variable](#) struct.

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>type</i>	Type of input file.

Definition at line 67 of file [variable.c](#).

```
00071 {
00072     #if DEBUG_VARIABLE
00073         fprintf (stderr, "variable_free: start\n");
00074     #endif
00075     if (type == INPUT_TYPE_XML)
00076         xmlFree (variable->name);
00077     else
00078         g_free (variable->name);
00079     #if DEBUG_VARIABLE
00080         fprintf (stderr, "variable_free: end\n");
00081     #endif
00082 }
```

#### 4.29.3.3 variable\_open\_json()

```
int variable_open_json (
    Variable * variable,
    JsonNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

##### Returns

1 on success, 0 on error.

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	Number of steps to do the hill climbing method.

Definition at line 270 of file [variable.c](#).

```
00275 {
00276     JsonObject *object;
00277     const char *label;
00278     int error_code;
00279     #if DEBUG_VARIABLE
00280         fprintf (stderr, "variable_open_json: start\n");
00281     #endif
00282     object = json_node_get_object (node);
00283     label = json_object_get_string_member (object, LABEL_NAME);
00284     if (!label)
```

```

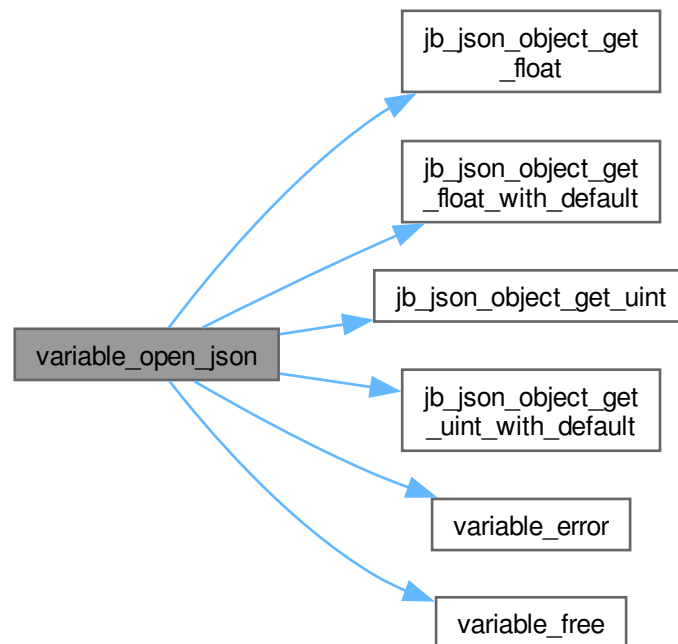
00285     {
00286         variable_error (variable, _("no name"));
00287         goto exit_on_error;
00288     }
00289     variable->name = g_strdup (label);
00290     if (json_object_get_member (object, LABEL_MINIMUM))
00291     {
00292         variable->rangemin
00293         = jb_json_object_get_float (object, LABEL_MINIMUM, &error_code);
00294         if (!error_code)
00295         {
00296             variable_error (variable, _("bad minimum"));
00297             goto exit_on_error;
00298         }
00299         variable->rangeminabs
00300         = jb_json_object_get_float_with_default (object, LABEL_ABSOLUTE_MINIMUM,
00301                                                 &error_code, -G_MAXDOUBLE);
00302         if (!error_code)
00303         {
00304             variable_error (variable, _("bad absolute minimum"));
00305             goto exit_on_error;
00306         }
00307         if (variable->rangemin < variable->rangeminabs)
00308         {
00309             variable_error (variable, _("minimum range not allowed"));
00310             goto exit_on_error;
00311         }
00312     }
00313     else
00314     {
00315         variable_error (variable, _("no minimum range"));
00316         goto exit_on_error;
00317     }
00318     if (json_object_get_member (object, LABEL_MAXIMUM))
00319     {
00320         variable->rangemax
00321         = jb_json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00322         if (!error_code)
00323         {
00324             variable_error (variable, _("bad maximum"));
00325             goto exit_on_error;
00326         }
00327         variable->rangemaxabs
00328         = jb_json_object_get_float_with_default (object, LABEL_ABSOLUTE_MAXIMUM,
00329                                                 &error_code, G_MAXDOUBLE);
00330         if (!error_code)
00331         {
00332             variable_error (variable, _("bad absolute maximum"));
00333             goto exit_on_error;
00334         }
00335         if (variable->rangemax > variable->rangemaxabs)
00336         {
00337             variable_error (variable, _("maximum range not allowed"));
00338             goto exit_on_error;
00339         }
00340         if (variable->rangemax < variable->rangemin)
00341         {
00342             variable_error (variable, _("bad range"));
00343             goto exit_on_error;
00344         }
00345     }
00346     else
00347     {
00348         variable_error (variable, _("no maximum range"));
00349         goto exit_on_error;
00350     }
00351     variable->precision
00352     = jb_json_object_get_uint_with_default (object, LABEL_PRECISION,
00353                                           &error_code, DEFAULT_PRECISION);
00354     if (!error_code || variable->precision >= NPRECISIONS)
00355     {
00356         variable_error (variable, _("bad precision"));
00357         goto exit_on_error;
00358     }
00359     if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)
00360     {
00361         if (json_object_get_member (object, LABEL_NSWEEPS))
00362         {
00363             variable->nsweeps
00364             = jb_json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00365             if (!error_code || !variable->nsweeps)
00366             {
00367                 variable_error (variable, _("bad sweeps"));
00368                 goto exit_on_error;
00369             }
00370         }
00371     }
00372     else

```

```
00372     {
00373         variable_error (variable, _("no sweeps number"));
00374         goto exit_on_error;
00375     }
00376 #if DEBUG_VARIABLE
00377     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00378 #endif
00379 }
00380 if (algorithm == ALGORITHM_GENETIC)
00381 {
00382     // Obtaining bits representing each variable
00383     if (json_object_get_member (object, LABEL_NBITS))
00384     {
00385         variable->nbits
00386         = jb_json_object_get_uint (object, LABEL_NBITS, &error_code);
00387         if (!error_code || !variable->nbits)
00388         {
00389             variable_error (variable, _("invalid bits number"));
00390             goto exit_on_error;
00391         }
00392     }
00393     else
00394     {
00395         variable_error (variable, _("no bits number"));
00396         goto exit_on_error;
00397     }
00398 }
00399 else if (nsteps)
00400 {
00401     variable->step
00402     = jb_json_object_get_float (object, LABEL_STEP, &error_code);
00403     if (!error_code || variable->step < 0.)
00404     {
00405         variable_error (variable, _("bad step size"));
00406         goto exit_on_error;
00407     }
00408 }
00409
00410 #if DEBUG_VARIABLE
00411     fprintf (stderr, "variable_open_json: end\n");
00412 #endif
00413     return 1;
00414 exit_on_error:
00415     variable_free (variable, INPUT_TYPE_JSON);
00416 #if DEBUG_VARIABLE
00417     fprintf (stderr, "variable_open_json: end\n");
00418 #endif
00419     return 0;
00420 }
```



Here is the call graph for this function:



#### 4.29.3.4 variable\_open\_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

##### Returns

1 on success, 0 on error.

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	Number of steps to do the hill climbing method.

Definition at line 107 of file [variable.c](#).

```

00112 {
00113     int error_code;
00114
00115     #if DEBUG_VARIABLE
00116     fprintf (stderr, "variable_open_xml: start\n");
00117     #endif
00118
00119     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00120     if (!variable->name)
00121     {
00122         variable_error (variable, _("no name"));
00123         goto exit_on_error;
00124     }
00125     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00126     {
00127         variable->rangemin
00128             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00129                                     &error_code);
00130         if (!error_code)
00131         {
00132             variable_error (variable, _("bad minimum"));
00133             goto exit_on_error;
00134         }
00135         variable->rangeminabs = jb_xml_node_get_float_with_default
00136             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, &error_code,
00137             -G_MAXDOUBLE);
00138         if (!error_code)
00139         {
00140             variable_error (variable, _("bad absolute minimum"));
00141             goto exit_on_error;
00142         }
00143         if (variable->rangemin < variable->rangeminabs)
00144         {
00145             variable_error (variable, _("minimum range not allowed"));
00146             goto exit_on_error;
00147         }
00148     }
00149     else
00150     {
00151         variable_error (variable, _("no minimum range"));
00152         goto exit_on_error;
00153     }
00154     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00155     {
00156         variable->rangemax
00157             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00158                                     &error_code);
00159         if (!error_code)
00160         {
00161             variable_error (variable, _("bad maximum"));
00162             goto exit_on_error;
00163         }
00164         variable->rangemaxabs = jb_xml_node_get_float_with_default
00165             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, &error_code,
00166             G_MAXDOUBLE);
00167         if (!error_code)
00168         {
00169             variable_error (variable, _("bad absolute maximum"));
00170             goto exit_on_error;
00171         }
00172         if (variable->rangemax > variable->rangemaxabs)
00173         {
00174             variable_error (variable, _("maximum range not allowed"));
00175             goto exit_on_error;
00176         }
00177         if (variable->rangemax < variable->rangemin)
00178         {
00179             variable_error (variable, _("bad range"));
00180             goto exit_on_error;
00181         }
00182     }
00183     else
00184     {
00185         variable_error (variable, _("no maximum range"));
00186         goto exit_on_error;
00187     }
00188     variable->precision
00189         = jb_xml_node_get_uint_with_default (node,
00190                                             (const xmlChar *) LABEL_PRECISION,
00191                                             &error_code, DEFAULT_PRECISION);
00192     if (!error_code || variable->precision >= NPRECISIONS)
00193     {
00194         variable_error (variable, _("bad precision"));
00195         goto exit_on_error;
00196     }
00197     if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)

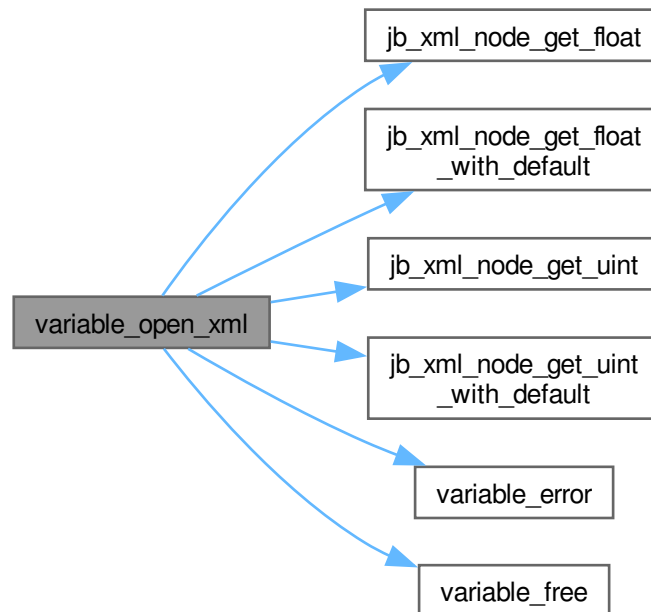
```

```

00199     {
00200         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00201         {
00202             variable->nsweeps
00203                 = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00204                                         &error_code);
00205             if (!error_code || !variable->nsweeps)
00206             {
00207                 variable_error (variable, _("bad sweeps"));
00208                 goto exit_on_error;
00209             }
00210         }
00211         else
00212         {
00213             variable_error (variable, _("no sweeps number"));
00214             goto exit_on_error;
00215         }
00216 #if DEBUG_VARIABLE
00217     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00218 #endif
00219     }
00220     if (algorithm == ALGORITHM_GENETIC)
00221     {
00222         // Obtaining bits representing each variable
00223         if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00224         {
00225             variable->nbits
00226                 = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00227                                         &error_code);
00228             if (!error_code || !variable->nbits)
00229             {
00230                 variable_error (variable, _("invalid bits number"));
00231                 goto exit_on_error;
00232             }
00233         }
00234         else
00235         {
00236             variable_error (variable, _("no bits number"));
00237             goto exit_on_error;
00238         }
00239     }
00240     else if (nsteps)
00241     {
00242         variable->step
00243             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_STEP,
00244                                     &error_code);
00245         if (!error_code || variable->step < 0.)
00246         {
00247             variable_error (variable, _("bad step size"));
00248             goto exit_on_error;
00249         }
00250     }
00251 #if DEBUG_VARIABLE
00252     fprintf (stderr, "variable_open_xml: end\n");
00253 #endif
00254     return 1;
00255 exit_on_error:
00256     variable_free (variable, INPUT_TYPE_XML);
00257 #if DEBUG_VARIABLE
00258     fprintf (stderr, "variable_open_xml: end\n");
00259 #endif
00260     return 0;
00261 }
00262 }

```

Here is the call graph for this function:



## 4.29.4 Variable Documentation

### 4.29.4.1 format

```
const char* format[NPRECISIONS]
```

**Initial value:**

```
= {
    "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
    "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
}
```

Array of C-strings with variable formats.

Definition at line 53 of file [variable.c](#).

```
00053 {
00054     "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
00055     "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
00056 };
```

### 4.29.4.2 precision

```
const double precision[NPRECISIONS]
```

**Initial value:**

```
= {
    1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
    1e-12, 1e-13, 1e-14
}
```

Array of variable precisions.

Definition at line 58 of file [variable.c](#).

```
00058 {
00059     1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00060     1e-12, 1e-13, 1e-14
00061 };
```

## 4.30 variable.c

[Go to the documentation of this file.](#)

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032  #define _GNU_SOURCE
00033  #include "config.h"
00034  #include <stdio.h>
00035  #include <libxml/parser.h>
00036  #include <libintl.h>
00037  #include <glib.h>
00038  #include <json-glib/json-glib.h>
00039  #include "jb/src/jb_xml.h"
00040  #include "jb/src/jb_json.h"
00041  #include "jb/src/jb_win.h"
00042  #include "tools.h"
00043  #include "variable.h"
00044
00045  #define DEBUG_VARIABLE 0
00046
00047  const char *format[NPRECISIONS] = {
00048      "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00049      "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00050  };
00051
00052  const double precision[NPRECISIONS] = {
00053      1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00054      1e-12, 1e-13, 1e-14
00055  };
00056
00057  void
00058  variable_free (Variable * variable,
00059               unsigned int type)
00060  {
00061      if (DEBUG_VARIABLE)
00062          fprintf (stderr, "variable_free: start\n");
00063      #endif
00064      if (type == INPUT_TYPE_XML)
00065          xmlFree (variable->name);
00066      else
00067          g_free (variable->name);
00068      #if DEBUG_VARIABLE
00069          fprintf (stderr, "variable_free: end\n");
00070      #endif
00071  }
00072
00073  void
00074  variable_error (Variable * variable,
00075                char *message)
00076  {
00077      char buffer[64];
00078      if (!variable->name)
00079          snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00080      else
00081          snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00082      error_message = g_strdup (buffer);
00083  }

```

```

00099 }
00100
00106 int
00107 variable_open_xml (Variable * variable,
00108                   xmlNode * node,
00109                   unsigned int algorithm,
00110                   unsigned int nsteps)
00112 {
00113     int error_code;
00114
00115     #if DEBUG_VARIABLE
00116     fprintf (stderr, "variable_open_xml: start\n");
00117     #endif
00118
00119     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00120     if (!variable->name)
00121     {
00122         variable_error (variable, _("no name"));
00123         goto exit_on_error;
00124     }
00125     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00126     {
00127         variable->rangemin
00128             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00129                                     &error_code);
00130         if (!error_code)
00131         {
00132             variable_error (variable, _("bad minimum"));
00133             goto exit_on_error;
00134         }
00135         variable->rangeminabs = jb_xml_node_get_float_with_default
00136             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, &error_code,
00137             -G_MAXDOUBLE);
00138         if (!error_code)
00139         {
00140             variable_error (variable, _("bad absolute minimum"));
00141             goto exit_on_error;
00142         }
00143         if (variable->rangemin < variable->rangeminabs)
00144         {
00145             variable_error (variable, _("minimum range not allowed"));
00146             goto exit_on_error;
00147         }
00148     }
00149     else
00150     {
00151         variable_error (variable, _("no minimum range"));
00152         goto exit_on_error;
00153     }
00154     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00155     {
00156         variable->rangemax
00157             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00158                                     &error_code);
00159         if (!error_code)
00160         {
00161             variable_error (variable, _("bad maximum"));
00162             goto exit_on_error;
00163         }
00164         variable->rangemaxabs = jb_xml_node_get_float_with_default
00165             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, &error_code,
00166             G_MAXDOUBLE);
00167         if (!error_code)
00168         {
00169             variable_error (variable, _("bad absolute maximum"));
00170             goto exit_on_error;
00171         }
00172         if (variable->rangemax > variable->rangemaxabs)
00173         {
00174             variable_error (variable, _("maximum range not allowed"));
00175             goto exit_on_error;
00176         }
00177         if (variable->rangemax < variable->rangemin)
00178         {
00179             variable_error (variable, _("bad range"));
00180             goto exit_on_error;
00181         }
00182     }
00183     else
00184     {
00185         variable_error (variable, _("no maximum range"));
00186         goto exit_on_error;
00187     }
00188     variable->precision
00189         = jb_xml_node_get_uint_with_default (node,
00190                                             (const xmlChar *) LABEL_PRECISION,

```

```

00192                                     &error_code, DEFAULT_PRECISION);
00193     if (!error_code || variable->precision >= NPRECISIONS)
00194     {
00195         variable_error (variable, _("bad precision"));
00196         goto exit_on_error;
00197     }
00198     if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)
00199     {
00200         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00201         {
00202             variable->nsweeps
00203             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00204                                     &error_code);
00205             if (!error_code || !variable->nsweeps)
00206             {
00207                 variable_error (variable, _("bad sweeps"));
00208                 goto exit_on_error;
00209             }
00210         }
00211         else
00212         {
00213             variable_error (variable, _("no sweeps number"));
00214             goto exit_on_error;
00215         }
00216 #if DEBUG_VARIABLE
00217         fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00218 #endif
00219     }
00220     if (algorithm == ALGORITHM_GENETIC)
00221     {
00222         // Obtaining bits representing each variable
00223         if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00224         {
00225             variable->nbits
00226             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00227                                     &error_code);
00228             if (!error_code || !variable->nbits)
00229             {
00230                 variable_error (variable, _("invalid bits number"));
00231                 goto exit_on_error;
00232             }
00233         }
00234         else
00235         {
00236             variable_error (variable, _("no bits number"));
00237             goto exit_on_error;
00238         }
00239     }
00240     else if (nsteps)
00241     {
00242         variable->step
00243         = jb_xml_node_get_float (node, (const xmlChar *) LABEL_STEP,
00244                                 &error_code);
00245         if (!error_code || variable->step < 0.)
00246         {
00247             variable_error (variable, _("bad step size"));
00248             goto exit_on_error;
00249         }
00250     }
00251 #if DEBUG_VARIABLE
00252     fprintf (stderr, "variable_open_xml: end\n");
00253 #endif
00254     return 1;
00255 exit_on_error:
00256     variable_free (variable, INPUT_TYPE_XML);
00257 #if DEBUG_VARIABLE
00258     fprintf (stderr, "variable_open_xml: end\n");
00259 #endif
00260     return 0;
00261 }
00262
00263 int
00264 variable_open_json (Variable * variable,
00265                    JsonNode * node,
00266                    unsigned int algorithm,
00267                    unsigned int nsteps)
00268 {
00269     JsonObject *object;
00270     const char *label;
00271     int error_code;
00272 #if DEBUG_VARIABLE
00273     fprintf (stderr, "variable_open_json: start\n");
00274 #endif
00275     object = json_node_get_object (node);
00276     label = json_object_get_string_member (object, LABEL_NAME);
00277     if (!label)

```

```

00285     {
00286         variable_error (variable, _("no name"));
00287         goto exit_on_error;
00288     }
00289     variable->name = g_strdup (label);
00290     if (json_object_get_member (object, LABEL_MINIMUM))
00291     {
00292         variable->rangemin
00293             = jb_json_object_get_float (object, LABEL_MINIMUM, &error_code);
00294         if (!error_code)
00295         {
00296             variable_error (variable, _("bad minimum"));
00297             goto exit_on_error;
00298         }
00299         variable->rangeminabs
00300             = jb_json_object_get_float_with_default (object, LABEL_ABSOLUTE_MINIMUM,
00301                                                         &error_code, -G_MAXDOUBLE);
00302         if (!error_code)
00303         {
00304             variable_error (variable, _("bad absolute minimum"));
00305             goto exit_on_error;
00306         }
00307         if (variable->rangemin < variable->rangeminabs)
00308         {
00309             variable_error (variable, _("minimum range not allowed"));
00310             goto exit_on_error;
00311         }
00312     }
00313     else
00314     {
00315         variable_error (variable, _("no minimum range"));
00316         goto exit_on_error;
00317     }
00318     if (json_object_get_member (object, LABEL_MAXIMUM))
00319     {
00320         variable->rangemax
00321             = jb_json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00322         if (!error_code)
00323         {
00324             variable_error (variable, _("bad maximum"));
00325             goto exit_on_error;
00326         }
00327         variable->rangemaxabs
00328             = jb_json_object_get_float_with_default (object, LABEL_ABSOLUTE_MAXIMUM,
00329                                                         &error_code, G_MAXDOUBLE);
00330         if (!error_code)
00331         {
00332             variable_error (variable, _("bad absolute maximum"));
00333             goto exit_on_error;
00334         }
00335         if (variable->rangemax > variable->rangemaxabs)
00336         {
00337             variable_error (variable, _("maximum range not allowed"));
00338             goto exit_on_error;
00339         }
00340         if (variable->rangemax < variable->rangemin)
00341         {
00342             variable_error (variable, _("bad range"));
00343             goto exit_on_error;
00344         }
00345     }
00346     else
00347     {
00348         variable_error (variable, _("no maximum range"));
00349         goto exit_on_error;
00350     }
00351     variable->precision
00352         = jb_json_object_get_uint_with_default (object, LABEL_PRECISION,
00353                                                 &error_code, DEFAULT_PRECISION);
00354     if (!error_code || variable->precision >= NPRECISIONS)
00355     {
00356         variable_error (variable, _("bad precision"));
00357         goto exit_on_error;
00358     }
00359     if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)
00360     {
00361         if (json_object_get_member (object, LABEL_NSWEEPS))
00362         {
00363             variable->nsweeps
00364                 = jb_json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00365             if (!error_code || !variable->nsweeps)
00366             {
00367                 variable_error (variable, _("bad sweeps"));
00368                 goto exit_on_error;
00369             }
00370         }
00371     }
00372     else

```



```

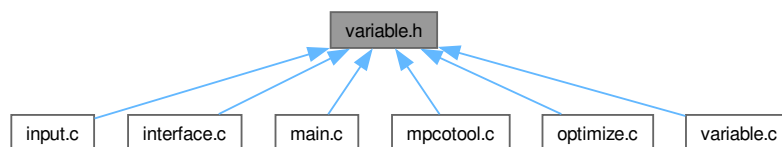
00372     {
00373         variable_error (variable, _("no sweeps number"));
00374         goto exit_on_error;
00375     }
00376 #if DEBUG_VARIABLE
00377     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00378 #endif
00379 }
00380 if (algorithm == ALGORITHM_GENETIC)
00381 {
00382     // Obtaining bits representing each variable
00383     if (json_object_get_member (object, LABEL_NBITS))
00384     {
00385         variable->nbits
00386         = jb_json_object_get_uint (object, LABEL_NBITS, &error_code);
00387         if (!error_code || !variable->nbits)
00388         {
00389             variable_error (variable, _("invalid bits number"));
00390             goto exit_on_error;
00391         }
00392     }
00393     else
00394     {
00395         variable_error (variable, _("no bits number"));
00396         goto exit_on_error;
00397     }
00398 }
00399 else if (nsteps)
00400 {
00401     variable->step
00402     = jb_json_object_get_float (object, LABEL_STEP, &error_code);
00403     if (!error_code || variable->step < 0.)
00404     {
00405         variable_error (variable, _("bad step size"));
00406         goto exit_on_error;
00407     }
00408 }
00409
00410 #if DEBUG_VARIABLE
00411     fprintf (stderr, "variable_open_json: end\n");
00412 #endif
00413     return 1;
00414 exit_on_error:
00415     variable_free (variable, INPUT_TYPE_JSON);
00416 #if DEBUG_VARIABLE
00417     fprintf (stderr, "variable_open_json: end\n");
00418 #endif
00419     return 0;
00420 }

```

## 4.31 variable.h File Reference

Header file to define the variable data.

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



### Data Structures

- struct [Variable](#)

*Struct to define the variable data.*

## Enumerations

- enum [Algorithm](#) { [ALGORITHM\\_MONTE\\_CARLO](#) = 0 , [ALGORITHM\\_SWEEP](#) = 1 , [ALGORITHM\\_GENETIC](#) = 2 , [ALGORITHM\\_ORTHOGONAL](#) = 3 }

*Enum to define the algorithms.*

## Functions

- void [variable\\_free](#) ([Variable](#) \*variable, unsigned int type)
- void [variable\\_error](#) ([Variable](#) \*variable, char \*message)
- int [variable\\_open\\_xml](#) ([Variable](#) \*variable, xmlDoc \*node, unsigned int algorithm, unsigned int nsteps)
- int [variable\\_open\\_json](#) ([Variable](#) \*variable, cJSON \*node, unsigned int algorithm, unsigned int nsteps)

## Variables

- const char \* [format](#) [[NPRECISIONS](#)]  
*Array of C-strings with variable formats.*
- const double [precision](#) [[NPRECISIONS](#)]  
*Array of variable precisions.*

### 4.31.1 Detailed Description

Header file to define the variable data.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2023, all rights reserved.

Definition in file [variable.h](#).

### 4.31.2 Enumeration Type Documentation

#### 4.31.2.1 Algorithm

enum [Algorithm](#)

Enum to define the algorithms.

#### Enumerator

<a href="#">ALGORITHM_MONTE_CARLO</a>	Monte-Carlo algorithm.
<a href="#">ALGORITHM_SWEEP</a>	Sweep algorithm.
<a href="#">ALGORITHM_GENETIC</a>	Genetic algorithm.
<a href="#">ALGORITHM_ORTHOGONAL</a>	Orthogonal sampling algorithm.

Definition at line 42 of file [variable.h](#).

```
00043 {
00044     ALGORITHM_MONTE_CARLO = 0,
00045     ALGORITHM_SWEEP = 1,
00046     ALGORITHM_GENETIC = 2,
00047     ALGORITHM_ORTHOGONAL = 3
00048 };
```

### 4.31.3 Function Documentation

#### 4.31.3.1 `variable_error()`

```
void variable_error (
    Variable * variable,
    char * message )
```

Function to print a message error opening an [Variable](#) struct.

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>message</i>	Error message.

Definition at line 88 of file [variable.c](#).

```
00092 {
00093     char buffer[64];
00094     if (!variable->name)
00095         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00096     else
00097         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00098     error_message = g_strdup (buffer);
00099 }
```

#### 4.31.3.2 `variable_free()`

```
void variable_free (
    Variable * variable,
    unsigned int type )
```

Function to free the memory of a [Variable](#) struct.

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>type</i>	Type of input file.

Definition at line 67 of file [variable.c](#).

```
00071 {
00072     #if DEBUG_VARIABLE
00073         fprintf (stderr, "variable_free: start\n");
00074     #endif
00075     if (type == INPUT_TYPE_XML)
00076         xmlFree (variable->name);
00077     else
00078         g_free (variable->name);
00079     #if DEBUG_VARIABLE
00080         fprintf (stderr, "variable_free: end\n");
00081     #endif
00082 }
```

### 4.31.3.3 variable\_open\_json()

```
int variable_open_json (
    Variable * variable,
    XmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

#### Returns

1 on success, 0 on error.

#### Parameters

<i>variable</i>	Variable struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	Number of steps to do the hill climbing method.

Definition at line 270 of file [variable.c](#).

```
00275 {
00276     JsonObject *object;
00277     const char *label;
00278     int error_code;
00279     #if DEBUG_VARIABLE
00280     fprintf (stderr, "variable_open_json: start\n");
00281     #endif
00282     object = json_node_get_object (node);
00283     label = json_object_get_string_member (object, LABEL_NAME);
00284     if (!label)
00285     {
00286         variable_error (variable, _("no name"));
00287         goto exit_on_error;
00288     }
00289     variable->name = g_strdup (label);
00290     if (json_object_get_member (object, LABEL_MINIMUM))
00291     {
00292         variable->rangemin
00293             = jb_json_object_get_float (object, LABEL_MINIMUM, &error_code);
00294         if (!error_code)
00295         {
00296             variable_error (variable, _("bad minimum"));
00297             goto exit_on_error;
00298         }
00299         variable->rangeminabs
00300             = jb_json_object_get_float_with_default (object, LABEL_ABSOLUTE_MINIMUM,
00301                                                         &error_code, -G_MAXDOUBLE);
00302         if (!error_code)
00303         {
00304             variable_error (variable, _("bad absolute minimum"));
00305             goto exit_on_error;
00306         }
00307         if (variable->rangemin < variable->rangeminabs)
00308         {
00309             variable_error (variable, _("minimum range not allowed"));
00310             goto exit_on_error;
00311         }
00312     }
00313     else
00314     {
00315         variable_error (variable, _("no minimum range"));
00316         goto exit_on_error;
00317     }
00318     if (json_object_get_member (object, LABEL_MAXIMUM))
00319     {
00320         variable->rangemax
00321             = jb_json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00322         if (!error_code)
00323         {
00324             variable_error (variable, _("bad maximum"));
00325         }
00326     }
00327 }
```

```

00325         goto exit_on_error;
00326     }
00327     variable->rangemaxabs
00328     = jb_json_object_get_float_with_default (object, LABEL_ABSOLUTE_MAXIMUM,
00329                                             &error_code, G_MAXDOUBLE);
00330     if (!error_code)
00331     {
00332         variable_error (variable, _("bad absolute maximum"));
00333         goto exit_on_error;
00334     }
00335     if (variable->rangemax > variable->rangemaxabs)
00336     {
00337         variable_error (variable, _("maximum range not allowed"));
00338         goto exit_on_error;
00339     }
00340     if (variable->rangemax < variable->rangemin)
00341     {
00342         variable_error (variable, _("bad range"));
00343         goto exit_on_error;
00344     }
00345 }
00346 else
00347 {
00348     variable_error (variable, _("no maximum range"));
00349     goto exit_on_error;
00350 }
00351 variable->precision
00352 = jb_json_object_get_uint_with_default (object, LABEL_PRECISION,
00353                                         &error_code, DEFAULT_PRECISION);
00354 if (!error_code || variable->precision >= NPRECISIONS)
00355 {
00356     variable_error (variable, _("bad precision"));
00357     goto exit_on_error;
00358 }
00359 if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)
00360 {
00361     if (json_object_get_member (object, LABEL_NSWEEPS))
00362     {
00363         variable->nsweeps
00364         = jb_json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00365         if (!error_code || !variable->nsweeps)
00366         {
00367             variable_error (variable, _("bad sweeps"));
00368             goto exit_on_error;
00369         }
00370     }
00371     else
00372     {
00373         variable_error (variable, _("no sweeps number"));
00374         goto exit_on_error;
00375     }
00376 #if DEBUG_VARIABLE
00377     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00378 #endif
00379 }
00380 if (algorithm == ALGORITHM_GENETIC)
00381 {
00382     // Obtaining bits representing each variable
00383     if (json_object_get_member (object, LABEL_NBITS))
00384     {
00385         variable->nbits
00386         = jb_json_object_get_uint (object, LABEL_NBITS, &error_code);
00387         if (!error_code || !variable->nbits)
00388         {
00389             variable_error (variable, _("invalid bits number"));
00390             goto exit_on_error;
00391         }
00392     }
00393     else
00394     {
00395         variable_error (variable, _("no bits number"));
00396         goto exit_on_error;
00397     }
00398 }
00399 else if (nsteps)
00400 {
00401     variable->step
00402     = jb_json_object_get_float (object, LABEL_STEP, &error_code);
00403     if (!error_code || variable->step < 0.)
00404     {
00405         variable_error (variable, _("bad step size"));
00406         goto exit_on_error;
00407     }
00408 }
00409 #if DEBUG_VARIABLE
00410 fprintf (stderr, "variable_open_json: end\n");

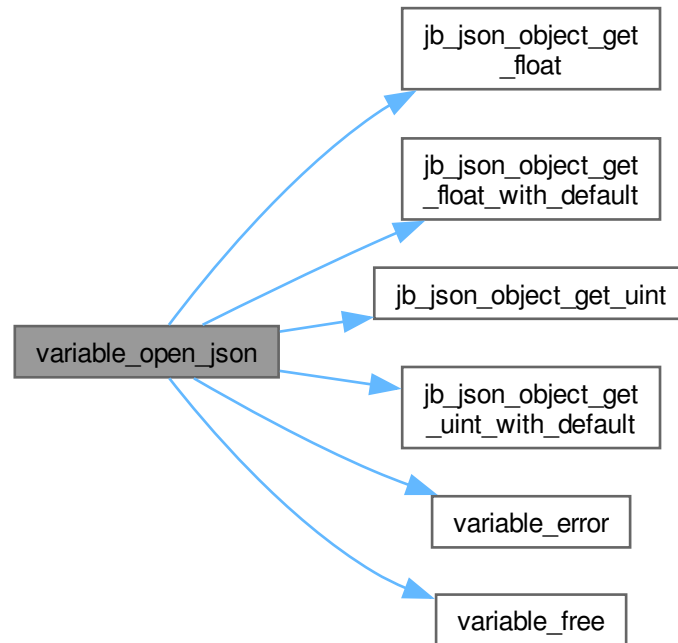
```

```

00412 #endif
00413     return 1;
00414 exit_on_error:
00415     variable_free (variable, INPUT_TYPE_JSON);
00416     #if DEBUG_VARIABLE
00417     fprintf (stderr, "variable_open_json: end\n");
00418     #endif
00419     return 0;
00420 }

```

Here is the call graph for this function:



#### 4.31.3.4 variable\_open\_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

##### Returns

1 on success, 0 on error.

##### Parameters

<i>variable</i>	Variable struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	Number of steps to do the hill climbing method.

Definition at line 107 of file [variable.c](#).

```

00112 {
00113     int error_code;
00114
00115     #if DEBUG_VARIABLE
00116     fprintf (stderr, "variable_open_xml: start\n");
00117     #endif
00118
00119     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00120     if (!variable->name)
00121     {
00122         variable_error (variable, _("no name"));
00123         goto exit_on_error;
00124     }
00125     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00126     {
00127         variable->rangemin
00128         = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00129                                 &error_code);
00130         if (!error_code)
00131         {
00132             variable_error (variable, _("bad minimum"));
00133             goto exit_on_error;
00134         }
00135         variable->rangeminabs = jb_xml_node_get_float_with_default
00136             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, &error_code,
00137             -G_MAXDOUBLE);
00138         if (!error_code)
00139         {
00140             variable_error (variable, _("bad absolute minimum"));
00141             goto exit_on_error;
00142         }
00143     }
00144     if (variable->rangemin < variable->rangeminabs)
00145     {
00146         variable_error (variable, _("minimum range not allowed"));
00147         goto exit_on_error;
00148     }
00149 }
00150 else
00151 {
00152     variable_error (variable, _("no minimum range"));
00153     goto exit_on_error;
00154 }
00155 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00156 {
00157     variable->rangemax
00158     = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00159                             &error_code);
00160     if (!error_code)
00161     {
00162         variable_error (variable, _("bad maximum"));
00163         goto exit_on_error;
00164     }
00165     variable->rangemaxabs = jb_xml_node_get_float_with_default
00166         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, &error_code,
00167         G_MAXDOUBLE);
00168     if (!error_code)
00169     {
00170         variable_error (variable, _("bad absolute maximum"));
00171         goto exit_on_error;
00172     }
00173     if (variable->rangemax > variable->rangemaxabs)
00174     {
00175         variable_error (variable, _("maximum range not allowed"));
00176         goto exit_on_error;
00177     }
00178     if (variable->rangemax < variable->rangemin)
00179     {
00180         variable_error (variable, _("bad range"));
00181         goto exit_on_error;
00182     }
00183 }
00184 else
00185 {
00186     variable_error (variable, _("no maximum range"));
00187     goto exit_on_error;
00188 }
00189 variable->precision
00190     = jb_xml_node_get_uint_with_default (node,
00191                                         (const xmlChar *) LABEL_PRECISION,
00192                                         &error_code, DEFAULT_PRECISION);
00193 if (!error_code || variable->precision >= NPRECISIONS)
00194 {
00195     variable_error (variable, _("bad precision"));
00196     goto exit_on_error;
00197 }

```

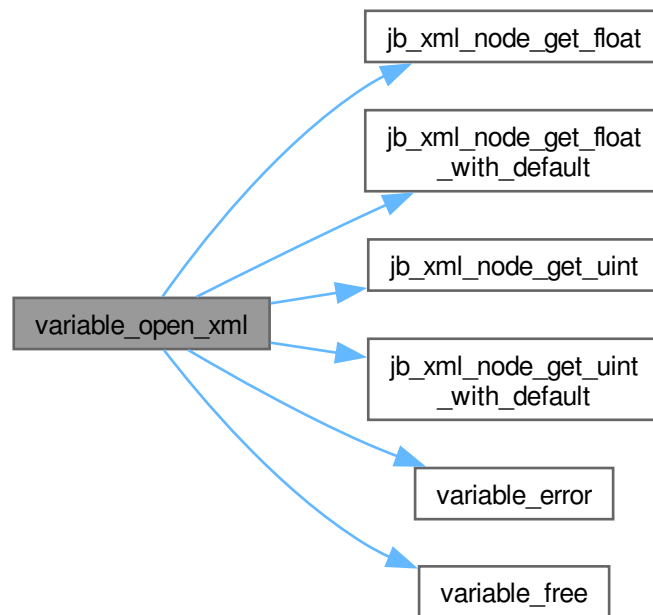
```

00198     if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)
00199     {
00200         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00201         {
00202             variable->nsweeps
00203             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00204                                     &error_code);
00205             if (!error_code || !variable->nsweeps)
00206             {
00207                 variable_error (variable, _("bad sweeps"));
00208                 goto exit_on_error;
00209             }
00210         }
00211         else
00212         {
00213             variable_error (variable, _("no sweeps number"));
00214             goto exit_on_error;
00215         }
00216 #if DEBUG_VARIABLE
00217         fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00218 #endif
00219     }
00220     if (algorithm == ALGORITHM_GENETIC)
00221     {
00222         // Obtaining bits representing each variable
00223         if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00224         {
00225             variable->nbits
00226             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00227                                     &error_code);
00228             if (!error_code || !variable->nbits)
00229             {
00230                 variable_error (variable, _("invalid bits number"));
00231                 goto exit_on_error;
00232             }
00233         }
00234         else
00235         {
00236             variable_error (variable, _("no bits number"));
00237             goto exit_on_error;
00238         }
00239     }
00240     else if (nsteps)
00241     {
00242         variable->step
00243         = jb_xml_node_get_float (node, (const xmlChar *) LABEL_STEP,
00244                                 &error_code);
00245         if (!error_code || variable->step < 0.)
00246         {
00247             variable_error (variable, _("bad step size"));
00248             goto exit_on_error;
00249         }
00250     }
00251 #if DEBUG_VARIABLE
00252     fprintf (stderr, "variable_open_xml: end\n");
00253 #endif
00254     return 1;
00255 exit_on_error:
00256     variable_free (variable, INPUT_TYPE_XML);
00257 #if DEBUG_VARIABLE
00258     fprintf (stderr, "variable_open_xml: end\n");
00259 #endif
00260     return 0;
00261 }

```



Here is the call graph for this function:



## 4.31.4 Variable Documentation

### 4.31.4.1 format

```
const char* format[NPRECISIONS] [extern]
```

Array of C-strings with variable formats.

Definition at line 53 of file [variable.c](#).

```

00053      {
00054          "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
00055          "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
00056      };

```

### 4.31.4.2 precision

```
const double precision[NPRECISIONS] [extern]
```

Array of variable precisions.

Definition at line 58 of file [variable.c](#).

```

00058      {
00059          1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00060          1e-12, 1e-13, 1e-14
00061      };

```

## 4.32 variable.h

[Go to the documentation of this file.](#)

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2023, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014      this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017      this list of conditions and the following disclaimer in the
00018      documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032 #ifndef VARIABLE__H
00033 #define VARIABLE__H 1
00034
00035 enum Algorithm
00036 {
00037     ALGORITHM_MONTE_CARLO = 0,
00038     ALGORITHM_SWEEP = 1,
00039     ALGORITHM_GENETIC = 2,
00040     ALGORITHM_ORTHOGONAL = 3
00041 };
00042
00043 typedef struct
00044 {
00045     char *name;
00046     double rangemin;
00047     double rangemax;
00048     double rangeminabs;
00049     double rangemaxabs;
00050     double step;
00051     unsigned int precision;
00052     unsigned int nsweeps;
00053     unsigned int nbits;
00054 } Variable;
00055
00056 extern const char *format[NPRECISIONS];
00057 extern const double precision[NPRECISIONS];
00058
00059 // Public functions
00060 void variable_free (Variable * variable, unsigned int type);
00061 void variable_error (Variable * variable, char *message);
00062 int variable_open_xml (Variable * variable, xmlNode * node,
00063                       unsigned int algorithm, unsigned int nsteps);
00064 int variable_open_json (Variable * variable, JsonNode * node,
00065                        unsigned int algorithm, unsigned int nsteps);
00066
00067 #endif

```

# Index

- adaptation\_ratio
  - Input, [8](#)
  - Optimize, [16](#)
- Algorithm
  - variable.h, [330](#)
- algorithm
  - Input, [8](#)
  - Optimize, [16](#)
- ALGORITHM\_GENETIC
  - variable.h, [330](#)
- ALGORITHM\_MONTE\_CARLO
  - variable.h, [330](#)
- ALGORITHM\_ORTHOGONAL
  - variable.h, [330](#)
- ALGORITHM\_SWEEP
  - variable.h, [330](#)
- application\_directory
  - Window, [37](#)
- box\_buttons
  - Window, [37](#)
- button\_about
  - Window, [37](#)
- button\_add\_experiment
  - Window, [37](#)
- button\_add\_variable
  - Window, [37](#)
- button\_algorithm
  - Window, [38](#)
- button\_cleaner
  - Window, [38](#)
- button\_climbing
  - Window, [38](#)
- button\_evaluator
  - Window, [38](#)
- button\_exit
  - Window, [38](#)
- button\_experiment
  - Window, [38](#)
- button\_help
  - Window, [39](#)
- button\_norm
  - Window, [39](#)
- button\_open
  - Window, [39](#)
- button\_options
  - Window, [39](#)
- button\_remove\_experiment
  - Window, [39](#)
- button\_remove\_variable
  - Window, [39](#)
- button\_run
  - Window, [40](#)
- button\_save
  - Window, [40](#)
- button\_simulator
  - Window, [40](#)
- button\_template
  - Window, [40](#)
- calculation\_time
  - Optimize, [16](#)
- check\_cleaner
  - Window, [40](#)
- check\_climbing
  - Window, [40](#)
- check\_evaluator
  - Window, [41](#)
- check\_maxabs
  - Window, [41](#)
- check\_minabs
  - Window, [41](#)
- check\_template
  - Window, [41](#)
- cleaner
  - Input, [8](#)
  - Optimize, [16](#)
- climbing
  - Input, [8](#)
  - Optimize, [17](#)
- CLIMBING\_METHOD\_COORDINATES
  - input.h, [126](#)
- CLIMBING\_METHOD\_RANDOM
  - input.h, [126](#)
- ClimbingMethod
  - input.h, [126](#)
- combo\_experiment
  - Window, [41](#)
- combo\_variable
  - Window, [41](#)
- config.h, [57](#)
  - DEFAULT\_PRECISION, [61](#)
  - DEFAULT\_RANDOM\_SEED, [61](#)
  - DEFAULT\_RELAXATION, [61](#)
  - INPUT\_TYPE, [72](#)
  - INPUT\_TYPE\_JSON, [73](#)
  - INPUT\_TYPE\_XML, [73](#)
  - LABEL\_ABSOLUTE\_MAXIMUM, [61](#)
  - LABEL\_ABSOLUTE\_MINIMUM, [61](#)
  - LABEL\_ADAPTATION, [61](#)

- LABEL\_ALGORITHM, [61](#)
- LABEL\_CLEANER, [62](#)
- LABEL\_CLIMBING, [62](#)
- LABEL\_COORDINATES, [62](#)
- LABEL\_EUCLIDIAN, [62](#)
- LABEL\_EVALUATOR, [62](#)
- LABEL\_EXPERIMENT, [62](#)
- LABEL\_EXPERIMENTS, [63](#)
- LABEL\_GENETIC, [63](#)
- LABEL\_INPUT1, [63](#)
- LABEL\_INPUT2, [63](#)
- LABEL\_INPUT3, [63](#)
- LABEL\_INPUT4, [63](#)
- LABEL\_INPUT5, [64](#)
- LABEL\_INPUT6, [64](#)
- LABEL\_INPUT7, [64](#)
- LABEL\_INPUT8, [64](#)
- LABEL\_MAXIMUM, [64](#)
- LABEL\_MINIMUM, [64](#)
- LABEL\_MONTE\_CARLO, [65](#)
- LABEL\_MUTATION, [65](#)
- LABEL\_NAME, [65](#)
- LABEL\_NBEST, [65](#)
- LABEL\_NBITS, [65](#)
- LABEL\_NESTIMATES, [65](#)
- LABEL\_NFINAL\_STEPS, [66](#)
- LABEL\_NGENERATIONS, [66](#)
- LABEL\_NITERATIONS, [66](#)
- LABEL\_NORM, [66](#)
- LABEL\_NPOPULATION, [66](#)
- LABEL\_NSIMULATIONS, [66](#)
- LABEL\_NSTEPS, [67](#)
- LABEL\_NSWEEPS, [67](#)
- LABEL\_OPTIMIZE, [67](#)
- LABEL\_ORTHOGONAL, [67](#)
- LABEL\_P, [67](#)
- LABEL\_PRECISION, [67](#)
- LABEL\_RANDOM, [68](#)
- LABEL\_RELAXATION, [68](#)
- LABEL\_REPRODUCTION, [68](#)
- LABEL\_RESULT\_FILE, [68](#)
- LABEL\_SEED, [68](#)
- LABEL\_SIMULATOR, [68](#)
- LABEL\_STEP, [69](#)
- LABEL\_SWEEP, [69](#)
- LABEL\_TAXICAB, [69](#)
- LABEL\_TEMPLATE1, [69](#)
- LABEL\_TEMPLATE2, [69](#)
- LABEL\_TEMPLATE3, [69](#)
- LABEL\_TEMPLATE4, [70](#)
- LABEL\_TEMPLATE5, [70](#)
- LABEL\_TEMPLATE6, [70](#)
- LABEL\_TEMPLATE7, [70](#)
- LABEL\_TEMPLATE8, [70](#)
- LABEL\_THRESHOLD, [70](#)
- LABEL\_TOLERANCE, [71](#)
- LABEL\_VARIABLE, [71](#)
- LABEL\_VARIABLES, [71](#)
- LABEL\_VARIABLES\_FILE, [71](#)
- LABEL\_WEIGHT, [71](#)
- LOCALE\_DIR, [71](#)
- MAX\_NINPUTS, [72](#)
- NALGORITHMS, [72](#)
- NCLIMBINGS, [72](#)
- NNORMS, [72](#)
- NPRECISIONS, [72](#)
- PROGRAM\_INTERFACE, [72](#)
- CP
  - optimize.c, [252](#)
- DEBUG\_EXPERIMENT
  - experiment.c, [76](#)
- DEBUG\_INPUT
  - input.c, [97](#)
- DEBUG\_INTERFACE
  - interface.c, [134](#)
- DEBUG\_MPCOTOOL
  - mpcotool.c, [241](#)
- DEBUG\_OPTIMIZE
  - optimize.c, [252](#)
- DEBUG\_VARIABLE
  - variable.c, [317](#)
- DEFAULT\_PRECISION
  - config.h, [61](#)
- DEFAULT\_RANDOM\_SEED
  - config.h, [61](#)
- DEFAULT\_RELAXATION
  - config.h, [61](#)
- dialog
  - Options, [26](#)
  - Running, [29](#)
- dialog\_cleaner
  - interface.c, [135](#)
- dialog\_cleaner\_close
  - interface.c, [135](#)
- dialog\_evaluator
  - interface.c, [136](#)
- dialog\_evaluator\_close
  - interface.c, [136](#)
- dialog\_name\_experiment\_close
  - interface.c, [137](#)
- dialog\_open\_close
  - interface.c, [138](#)
- dialog\_options\_close
  - interface.c, [139](#)
- dialog\_save\_close
  - interface.c, [139](#)
- dialog\_simulator
  - interface.c, [141](#)
- dialog\_simulator\_close
  - interface.c, [142](#)
- directory
  - Input, [9](#)
- entry\_result
  - Window, [42](#)
- entry\_variable

- Window, 42
- entry\_variables
  - Window, 42
- error\_best
  - Optimize, 17
- error\_message
  - tools.c, 312
  - tools.h, 315
- ERROR\_NORM\_EUCLIDIAN
  - input.h, 127
- ERROR\_NORM\_MAXIMUM
  - input.h, 127
- ERROR\_NORM\_P
  - input.h, 127
- ERROR\_NORM\_TAXICAB
  - input.h, 127
- error\_old
  - Optimize, 17
- ERROR\_TYPE
  - tools.h, 314
- ErrorNorm
  - input.h, 126
- evaluator
  - Input, 9
  - Optimize, 17
- Experiment, 5
  - name, 5
  - ninputs, 5
  - stencil, 6
  - template\_flags, 6
  - weight, 6
- experiment
  - Input, 9
  - Optimize, 17
  - Window, 42
- experiment.c, 74
  - DEBUG\_EXPERIMENT, 76
  - experiment\_error, 76
  - experiment\_free, 76
  - experiment\_new, 77
  - experiment\_open\_json, 77
  - experiment\_open\_xml, 79
  - stencil, 82
  - stencilbin, 82
- experiment.h, 87
  - experiment\_error, 88
  - experiment\_free, 88
  - experiment\_open\_json, 90
  - experiment\_open\_xml, 92
  - stencil, 95
  - stencilbin, 95
- experiment\_error
  - experiment.c, 76
  - experiment.h, 88
- experiment\_free
  - experiment.c, 76
  - experiment.h, 88
- experiment\_new
  - experiment.c, 77
  - experiment.h, 90
- experiment\_open\_json
  - experiment.c, 77
  - experiment.h, 90
- experiment\_open\_xml
  - experiment.c, 79
  - experiment.h, 92
- file
  - Optimize, 17
- file\_result
  - Optimize, 18
- file\_variables
  - Optimize, 18
- format
  - variable.c, 324
  - variable.h, 337
- frame\_algorithm
  - Window, 42
- frame\_experiment
  - Window, 42
- frame\_norm
  - Window, 43
- frame\_variable
  - Window, 43
- genetic\_variable
  - Optimize, 18
- grid
  - Options, 26
  - Running, 29
  - Window, 43
- grid\_algorithm
  - Window, 43
- grid\_climbing
  - Window, 43
- grid\_experiment
  - Window, 43
- grid\_files
  - Window, 44
- grid\_norm
  - Window, 44
- grid\_variable
  - Window, 44
- id\_experiment
  - Window, 44
- id\_experiment\_name
  - Window, 44
- id\_input
  - Window, 44
- id\_template
  - Window, 45
- id\_variable
  - Window, 45
- id\_variable\_label
  - Window, 45
- INFO\_TYPE
  - tools.h, 314

- Input, 6
  - adaptation\_ratio, 8
  - algorithm, 8
  - cleaner, 8
  - climbing, 8
  - directory, 9
  - evaluator, 9
  - experiment, 9
  - mutation\_ratio, 9
  - name, 9
  - nbest, 9
  - nestimates, 10
  - nexperiments, 10
  - nfinal\_steps, 10
  - niterations, 10
  - norm, 10
  - nsimulations, 10
  - nsteps, 11
  - nvariables, 11
  - p, 11
  - relaxation, 11
  - reproduction\_ratio, 11
  - result, 11
  - seed, 12
  - simulator, 12
  - template\_flags, 12
  - threshold, 12
  - tolerance, 12
  - type, 12
  - variable, 13
  - variables, 13
- input
  - input.c, 112
  - input.h, 130
- input.c, 96
  - DEBUG\_INPUT, 97
  - input, 112
  - input\_error, 97
  - input\_free, 98
  - input\_new, 98
  - input\_open, 99
  - input\_open\_json, 100
  - input\_open\_xml, 106
  - result\_name, 112
  - variables\_name, 112
- input.h, 125
  - CLIMBING\_METHOD\_COORDINATES, 126
  - CLIMBING\_METHOD\_RANDOM, 126
  - ClimbingMethod, 126
  - ERROR\_NORM\_EUCLIDIAN, 127
  - ERROR\_NORM\_MAXIMUM, 127
  - ERROR\_NORM\_P, 127
  - ERROR\_NORM\_TAXICAB, 127
  - ErrorNorm, 126
  - input, 130
  - input\_free, 127
  - input\_new, 128
  - input\_open, 128
  - result\_name, 130
  - variables\_name, 130
- input\_error
  - input.c, 97
- INPUT\_FILE
  - interface.c, 134
- input\_free
  - input.c, 98
  - input.h, 127
- input\_new
  - input.c, 98
  - input.h, 128
- input\_open
  - input.c, 99
  - input.h, 128
- input\_open\_json
  - input.c, 100
- input\_open\_xml
  - input.c, 106
- input\_save
  - interface.c, 142
- input\_save\_climbing\_json
  - interface.c, 143
- input\_save\_climbing\_xml
  - interface.c, 144
- input\_save\_json
  - interface.c, 145
- input\_save\_xml
  - interface.c, 148
- INPUT\_TYPE
  - config.h, 72
- INPUT\_TYPE\_JSON
  - config.h, 73
- INPUT\_TYPE\_XML
  - config.h, 73
- interface.c, 132
  - DEBUG\_INTERFACE, 134
  - dialog\_cleaner, 135
  - dialog\_cleaner\_close, 135
  - dialog\_evaluator, 136
  - dialog\_evaluator\_close, 136
  - dialog\_name\_experiment\_close, 137
  - dialog\_open\_close, 138
  - dialog\_options\_close, 139
  - dialog\_save\_close, 139
  - dialog\_simulator, 141
  - dialog\_simulator\_close, 142
  - INPUT\_FILE, 134
  - input\_save, 142
  - input\_save\_climbing\_json, 143
  - input\_save\_climbing\_xml, 144
  - input\_save\_json, 145
  - input\_save\_xml, 148
  - logo, 185
  - options, 186
  - options\_new, 150
  - running, 186
  - running\_new, 151

- window, 186
- window\_about, 152
- window\_add\_experiment, 152
- window\_add\_variable, 153
- window\_get\_algorithm, 154
- window\_get\_climbing, 154
- window\_get\_norm, 155
- window\_help, 156
- window\_inputs\_experiment, 156
- window\_label\_variable, 157
- window\_name\_experiment, 157
- window\_new, 158
- window\_open, 168
- window\_precision\_variable, 169
- window\_rangemax\_variable, 170
- window\_rangemaxabs\_variable, 170
- window\_rangemin\_variable, 170
- window\_rangeminabs\_variable, 170
- window\_read, 171
- window\_remove\_experiment, 173
- window\_remove\_variable, 174
- window\_run, 174
- window\_save, 175
- window\_save\_climbing, 176
- window\_set\_algorithm, 177
- window\_set\_experiment, 178
- window\_set\_variable, 178
- window\_step\_variable, 179
- window\_template\_experiment, 180
- window\_template\_experiment\_close, 180
- window\_update, 181
- window\_update\_climbing, 183
- window\_update\_variable, 184
- window\_weight\_experiment, 185
- interface.h, 221
  - MAX\_LENGTH, 222
  - window, 233
  - window\_new, 222
- JBW
  - main.c, 237
- label
  - Optimize, 18
  - Running, 29
- LABEL\_ABSOLUTE\_MAXIMUM
  - config.h, 61
- LABEL\_ABSOLUTE\_MINIMUM
  - config.h, 61
- LABEL\_ADAPTATION
  - config.h, 61
- label\_adaptation
  - Window, 45
- LABEL\_ALGORITHM
  - config.h, 61
- label\_bests
  - Window, 45
- label\_bits
  - Window, 45
- LABEL\_CLEANER
  - config.h, 62
- LABEL\_CLIMBING
  - config.h, 62
- label\_climbing
  - Options, 26
- LABEL\_COORDINATES
  - config.h, 62
- label\_estimates
  - Window, 46
- LABEL\_EUCLIDIAN
  - config.h, 62
- LABEL\_EVALUATOR
  - config.h, 62
- LABEL\_EXPERIMENT
  - config.h, 62
- label\_experiment
  - Window, 46
- LABEL\_EXPERIMENTS
  - config.h, 63
- label\_final\_steps
  - Window, 46
- label\_generations
  - Window, 46
- LABEL\_GENETIC
  - config.h, 63
- LABEL\_INPUT1
  - config.h, 63
- LABEL\_INPUT2
  - config.h, 63
- LABEL\_INPUT3
  - config.h, 63
- LABEL\_INPUT4
  - config.h, 63
- LABEL\_INPUT5
  - config.h, 64
- LABEL\_INPUT6
  - config.h, 64
- LABEL\_INPUT7
  - config.h, 64
- LABEL\_INPUT8
  - config.h, 64
- label\_iterations
  - Window, 46
- label\_max
  - Window, 46
- LABEL\_MAXIMUM
  - config.h, 64
- label\_min
  - Window, 47
- LABEL\_MINIMUM
  - config.h, 64
- LABEL\_MONTE\_CARLO
  - config.h, 65
- LABEL\_MUTATION
  - config.h, 65
- label\_mutation
  - Window, 47

LABEL\_NAME  
    config.h, 65

LABEL\_NBEST  
    config.h, 65

LABEL\_NBITS  
    config.h, 65

LABEL\_NESTIMATES  
    config.h, 65

LABEL\_NFINAL\_STEPS  
    config.h, 66

LABEL\_NGENERATIONS  
    config.h, 66

LABEL\_NITERATIONS  
    config.h, 66

LABEL\_NORM  
    config.h, 66

LABEL\_NPOPULATION  
    config.h, 66

LABEL\_NSIMULATIONS  
    config.h, 66

LABEL\_NSTEPS  
    config.h, 67

LABEL\_NSWEEPS  
    config.h, 67

LABEL\_OPTIMIZE  
    config.h, 67

LABEL\_ORTHOGONAL  
    config.h, 67

LABEL\_P  
    config.h, 67

label\_p  
    Window, 47

label\_population  
    Window, 47

LABEL\_PRECISION  
    config.h, 67

label\_precision  
    Window, 47

LABEL\_RANDOM  
    config.h, 68

LABEL\_RELAXATION  
    config.h, 68

label\_relaxation  
    Window, 47

LABEL\_REPRODUCTION  
    config.h, 68

label\_reproduction  
    Window, 48

label\_result  
    Window, 48

LABEL\_RESULT\_FILE  
    config.h, 68

LABEL\_SEED  
    config.h, 68

label\_seed  
    Options, 26

label\_simulations  
    Window, 48

LABEL\_SIMULATOR  
    config.h, 68

label\_simulator  
    Window, 48

LABEL\_STEP  
    config.h, 69

label\_step  
    Window, 48

label\_steps  
    Window, 48

LABEL\_SWEEP  
    config.h, 69

label\_sweeps  
    Window, 49

LABEL\_TAXICAB  
    config.h, 69

LABEL\_TEMPLATE1  
    config.h, 69

LABEL\_TEMPLATE2  
    config.h, 69

LABEL\_TEMPLATE3  
    config.h, 69

LABEL\_TEMPLATE4  
    config.h, 70

LABEL\_TEMPLATE5  
    config.h, 70

LABEL\_TEMPLATE6  
    config.h, 70

LABEL\_TEMPLATE7  
    config.h, 70

LABEL\_TEMPLATE8  
    config.h, 70

label\_threads  
    Options, 27

LABEL\_THRESHOLD  
    config.h, 70

label\_threshold  
    Window, 49

LABEL\_TOLERANCE  
    config.h, 71

label\_tolerance  
    Window, 49

LABEL\_VARIABLE  
    config.h, 71

label\_variable  
    Window, 49

LABEL\_VARIABLES  
    config.h, 71

label\_variables  
    Window, 49

LABEL\_VARIABLES\_FILE  
    config.h, 71

LABEL\_WEIGHT  
    config.h, 71

label\_weight  
    Window, 49

LOCALE\_DIR  
    config.h, 71



- logo
  - interface.c, 185
  - Window, 50
- main
  - main.c, 237
- main.c, 236
  - JBW, 237
  - main, 237
- main\_window
  - tools.c, 312
  - tools.h, 315
- MAX\_LENGTH
  - interface.h, 222
- MAX\_NINPUTS
  - config.h, 72
- mpcotool
  - mpcotool.c, 241
  - mpcotool.h, 247
- mpcotool.c, 239
  - DEBUG\_MPCOTOOL, 241
  - mpcotool, 241
- mpcotool.h, 246
  - mpcotool, 247
- mpi\_rank
  - Optimize, 18
- mutation\_ratio
  - Input, 9
  - Optimize, 18
- NALGORITHMS
  - config.h, 72
- name
  - Experiment, 5
  - Input, 9
  - Variable, 30
- nbest
  - Input, 9
  - Optimize, 19
- nbits
  - Optimize, 19
  - Variable, 30
- NCLIMBINGS
  - config.h, 72
- nend
  - Optimize, 19
- nend\_climbing
  - Optimize, 19
- nestimates
  - Input, 10
  - Optimize, 19
- nexperiments
  - Input, 10
  - Optimize, 19
  - Window, 50
- nfina\_steps
  - Input, 10
  - Optimize, 20
- ninputs
  - Experiment, 5
  - Optimize, 20
- niterations
  - Input, 10
  - Optimize, 20
- NNORMS
  - config.h, 72
- norm
  - Input, 10
- NPRECISIONS
  - config.h, 72
- nsaveds
  - Optimize, 20
- nsimulations
  - Input, 10
  - Optimize, 20
- nstart
  - Optimize, 20
- nstart\_climbing
  - Optimize, 21
- nsteps
  - Input, 11
  - Optimize, 21
- nsweeps
  - Optimize, 21
  - Variable, 30
- nthreads\_climbing
  - optimize.c, 283
  - optimize.h, 309
- nvariables
  - Input, 11
  - Optimize, 21
  - Window, 50
- Optimize, 13
  - adaptation\_ratio, 16
  - algorithm, 16
  - calculation\_time, 16
  - cleaner, 16
  - climbing, 17
  - error\_best, 17
  - error\_old, 17
  - evaluator, 17
  - experiment, 17
  - file, 17
  - file\_result, 18
  - file\_variables, 18
  - genetic\_variable, 18
  - label, 18
  - mpi\_rank, 18
  - mutation\_ratio, 18
  - nbest, 19
  - nbits, 19
  - nend, 19
  - nend\_climbing, 19
  - nestimates, 19
  - nexperiments, 19
  - nfina\_steps, 20
  - ninputs, 20

- iterations, 20
- nsaveds, 20
- nsimulations, 20
- nstart, 20
- nstart\_climbing, 21
- nsteps, 21
- nsweeps, 21
- nvariables, 21
- p, 21
- precision, 21
- rangemax, 22
- rangemaxabs, 22
- rangemin, 22
- rangeminabs, 22
- relaxation, 22
- reproduction\_ratio, 22
- result, 23
- rng, 23
- seed, 23
- simulation\_best, 23
- simulator, 23
- step, 23
- stop, 24
- template\_flags, 24
- thread, 24
- thread\_climbing, 24
- threshold, 24
- tolerance, 24
- value, 25
- value\_old, 25
- variables, 25
- weight, 25
- optimize
  - optimize.c, 283
  - optimize.h, 309
- optimize.c, 250
  - CP, 252
  - DEBUG\_OPTIMIZE, 252
  - nthreads\_climbing, 283
  - optimize, 283
  - optimize\_algorithm, 283
  - optimize\_best, 252
  - optimize\_best\_climbing, 253
  - optimize\_climbing, 254
  - optimize\_climbing\_best, 255
  - optimize\_climbing\_sequential, 255
  - optimize\_climbing\_thread, 256
  - optimize\_estimate\_climbing, 284
  - optimize\_estimate\_climbing\_coordinates, 257
  - optimize\_estimate\_climbing\_random, 257
  - optimize\_free, 258
  - optimize\_genetic, 258
  - optimize\_genetic\_objective, 259
  - optimize\_input, 260
  - optimize\_iterate, 261
  - optimize\_merge, 262
  - optimize\_merge\_old, 263
  - optimize\_MonteCarlo, 264
  - optimize\_norm, 284
  - optimize\_norm\_euclidian, 264
  - optimize\_norm\_maximum, 265
  - optimize\_norm\_p, 266
  - optimize\_norm\_taxicab, 267
  - optimize\_open, 268
  - optimize\_orthogonal, 272
  - optimize\_parse, 273
  - optimize\_print, 275
  - optimize\_refine, 275
  - optimize\_save\_old, 276
  - optimize\_save\_optimal, 277
  - optimize\_save\_variables, 277
  - optimize\_sequential, 278
  - optimize\_step, 279
  - optimize\_step\_climbing, 279
  - optimize\_sweep, 280
  - optimize\_synchronise, 281
  - optimize\_thread, 282
  - RM, 252
- optimize.h, 304
  - nthreads\_climbing, 309
  - optimize, 309
  - optimize\_free, 305
  - optimize\_open, 305
- optimize\_algorithm
  - optimize.c, 283
- optimize\_best
  - optimize.c, 252
- optimize\_best\_climbing
  - optimize.c, 253
- optimize\_climbing
  - optimize.c, 254
- optimize\_climbing\_best
  - optimize.c, 255
- optimize\_climbing\_sequential
  - optimize.c, 255
- optimize\_climbing\_thread
  - optimize.c, 256
- optimize\_estimate\_climbing
  - optimize.c, 284
- optimize\_estimate\_climbing\_coordinates
  - optimize.c, 257
- optimize\_estimate\_climbing\_random
  - optimize.c, 257
- optimize\_free
  - optimize.c, 258
  - optimize.h, 305
- optimize\_genetic
  - optimize.c, 258
- optimize\_genetic\_objective
  - optimize.c, 259
- optimize\_input
  - optimize.c, 260
- optimize\_iterate
  - optimize.c, 261
- optimize\_merge
  - optimize.c, 262

- optimize\_merge\_old
  - optimize.c, [263](#)
- optimize\_MonteCarlo
  - optimize.c, [264](#)
- optimize\_norm
  - optimize.c, [284](#)
- optimize\_norm\_euclidian
  - optimize.c, [264](#)
- optimize\_norm\_maximum
  - optimize.c, [265](#)
- optimize\_norm\_p
  - optimize.c, [266](#)
- optimize\_norm\_taxicab
  - optimize.c, [267](#)
- optimize\_open
  - optimize.c, [268](#)
  - optimize.h, [305](#)
- optimize\_orthogonal
  - optimize.c, [272](#)
- optimize\_parse
  - optimize.c, [273](#)
- optimize\_print
  - optimize.c, [275](#)
- optimize\_refine
  - optimize.c, [275](#)
- optimize\_save\_old
  - optimize.c, [276](#)
- optimize\_save\_optimal
  - optimize.c, [277](#)
- optimize\_save\_variables
  - optimize.c, [277](#)
- optimize\_sequential
  - optimize.c, [278](#)
- optimize\_step
  - optimize.c, [279](#)
- optimize\_step\_climbing
  - optimize.c, [279](#)
- optimize\_sweep
  - optimize.c, [280](#)
- optimize\_synchronise
  - optimize.c, [281](#)
- optimize\_thread
  - optimize.c, [282](#)
- Options, [25](#)
  - dialog, [26](#)
  - grid, [26](#)
  - label\_climbing, [26](#)
  - label\_seed, [26](#)
  - label\_threads, [27](#)
  - spin\_climbing, [27](#)
  - spin\_seed, [27](#)
  - spin\_threads, [27](#)
- options
  - interface.c, [186](#)
- options\_new
  - interface.c, [150](#)
- p
  - Input, [11](#)
- Optimize, [21](#)
- ParallelData, [27](#)
  - thread, [28](#)
- precision
  - Optimize, [21](#)
  - Variable, [31](#)
  - variable.c, [324](#)
  - variable.h, [337](#)
- PROGRAM\_INTERFACE
  - config.h, [72](#)
- rangemax
  - Optimize, [22](#)
  - Variable, [31](#)
- rangemaxabs
  - Optimize, [22](#)
  - Variable, [31](#)
- rangemin
  - Optimize, [22](#)
  - Variable, [31](#)
- rangeminabs
  - Optimize, [22](#)
  - Variable, [31](#)
- relaxation
  - Input, [11](#)
  - Optimize, [22](#)
- reproduction\_ratio
  - Input, [11](#)
  - Optimize, [22](#)
- result
  - Input, [11](#)
  - Optimize, [23](#)
- result\_name
  - input.c, [112](#)
  - input.h, [130](#)
- RM
  - optimize.c, [252](#)
- rng
  - Optimize, [23](#)
- Running, [28](#)
  - dialog, [29](#)
  - grid, [29](#)
  - label, [29](#)
  - spinner, [29](#)
- running
  - interface.c, [186](#)
- running\_new
  - interface.c, [151](#)
- scrolled\_max
  - Window, [50](#)
- scrolled\_maxabs
  - Window, [50](#)
- scrolled\_min
  - Window, [50](#)
- scrolled\_minabs
  - Window, [51](#)
- scrolled\_p
  - Window, [51](#)

- scrolled\_step
  - Window, [51](#)
- scrolled\_threshold
  - Window, [51](#)
- seed
  - Input, [12](#)
  - Optimize, [23](#)
- show\_pending
  - tools.c, [312](#)
  - tools.h, [315](#)
- simulation\_best
  - Optimize, [23](#)
- simulator
  - Input, [12](#)
  - Optimize, [23](#)
- spin\_adaptation
  - Window, [51](#)
- spin\_bests
  - Window, [51](#)
- spin\_bits
  - Window, [52](#)
- spin\_climbing
  - Options, [27](#)
- spin\_estimates
  - Window, [52](#)
- spin\_final\_steps
  - Window, [52](#)
- spin\_generations
  - Window, [52](#)
- spin\_iterations
  - Window, [52](#)
- spin\_max
  - Window, [52](#)
- spin\_maxabs
  - Window, [53](#)
- spin\_min
  - Window, [53](#)
- spin\_minabs
  - Window, [53](#)
- spin\_mutation
  - Window, [53](#)
- spin\_p
  - Window, [53](#)
- spin\_population
  - Window, [53](#)
- spin\_precision
  - Window, [54](#)
- spin\_relaxation
  - Window, [54](#)
- spin\_reproduction
  - Window, [54](#)
- spin\_seed
  - Options, [27](#)
- spin\_simulations
  - Window, [54](#)
- spin\_step
  - Window, [54](#)
- spin\_steps
  - Window, [54](#)
- spin\_sweeps
  - Window, [55](#)
- spin\_threads
  - Options, [27](#)
- spin\_threshold
  - Window, [55](#)
- spin\_tolerance
  - Window, [55](#)
- spin\_weight
  - Window, [55](#)
- spinner
  - Running, [29](#)
- stencil
  - Experiment, [6](#)
  - experiment.c, [82](#)
  - experiment.h, [95](#)
- stencilbin
  - experiment.c, [82](#)
  - experiment.h, [95](#)
- step
  - Optimize, [23](#)
  - Variable, [31](#)
- stop
  - Optimize, [24](#)
- template\_flags
  - Experiment, [6](#)
  - Input, [12](#)
  - Optimize, [24](#)
- thread
  - Optimize, [24](#)
  - ParallelData, [28](#)
- thread\_climbing
  - Optimize, [24](#)
- threshold
  - Input, [12](#)
  - Optimize, [24](#)
- tolerance
  - Input, [12](#)
  - Optimize, [24](#)
- tools.c, [311](#)
  - error\_message, [312](#)
  - main\_window, [312](#)
  - show\_pending, [312](#)
- tools.h, [313](#)
  - error\_message, [315](#)
  - ERROR\_TYPE, [314](#)
  - INFO\_TYPE, [314](#)
  - main\_window, [315](#)
  - show\_pending, [315](#)
- type
  - Input, [12](#)
- value
  - Optimize, [25](#)
- value\_old
  - Optimize, [25](#)
- Variable, [29](#)

- name, 30
- nbits, 30
- nsweeps, 30
- precision, 31
- rangemax, 31
- rangemaxabs, 31
- rangemin, 31
- rangeminabs, 31
- step, 31
- variable
  - Input, 13
  - Window, 55
- variable.c, 316
  - DEBUG\_VARIABLE, 317
  - format, 324
  - precision, 324
  - variable\_error, 317
  - variable\_free, 317
  - variable\_open\_json, 318
  - variable\_open\_xml, 321
- variable.h, 329
  - Algorithm, 330
  - ALGORITHM\_GENETIC, 330
  - ALGORITHM\_MONTE\_CARLO, 330
  - ALGORITHM\_ORTHOGONAL, 330
  - ALGORITHM\_SWEEP, 330
  - format, 337
  - precision, 337
  - variable\_error, 331
  - variable\_free, 331
  - variable\_open\_json, 331
  - variable\_open\_xml, 334
- variable\_error
  - variable.c, 317
  - variable.h, 331
- variable\_free
  - variable.c, 317
  - variable.h, 331
- variable\_open\_json
  - variable.c, 318
  - variable.h, 331
- variable\_open\_xml
  - variable.c, 321
  - variable.h, 334
- variables
  - Input, 13
  - Optimize, 25
- variables\_name
  - input.c, 112
  - input.h, 130
- weight
  - Experiment, 6
  - Optimize, 25
- Window, 32
  - application\_directory, 37
  - box\_buttons, 37
  - button\_about, 37
  - button\_add\_experiment, 37
  - button\_add\_variable, 37
  - button\_algorithm, 38
  - button\_cleaner, 38
  - button\_climbing, 38
  - button\_evaluator, 38
  - button\_exit, 38
  - button\_experiment, 38
  - button\_help, 39
  - button\_norm, 39
  - button\_open, 39
  - button\_options, 39
  - button\_remove\_experiment, 39
  - button\_remove\_variable, 39
  - button\_run, 40
  - button\_save, 40
  - button\_simulator, 40
  - button\_template, 40
  - check\_cleaner, 40
  - check\_climbing, 40
  - check\_evaluator, 41
  - check\_maxabs, 41
  - check\_minabs, 41
  - check\_template, 41
  - combo\_experiment, 41
  - combo\_variable, 41
  - entry\_result, 42
  - entry\_variable, 42
  - entry\_variables, 42
  - experiment, 42
  - frame\_algorithm, 42
  - frame\_experiment, 42
  - frame\_norm, 43
  - frame\_variable, 43
  - grid, 43
  - grid\_algorithm, 43
  - grid\_climbing, 43
  - grid\_experiment, 43
  - grid\_files, 44
  - grid\_norm, 44
  - grid\_variable, 44
  - id\_experiment, 44
  - id\_experiment\_name, 44
  - id\_input, 44
  - id\_template, 45
  - id\_variable, 45
  - id\_variable\_label, 45
  - label\_adaptation, 45
  - label\_bests, 45
  - label\_bits, 45
  - label\_estimates, 46
  - label\_experiment, 46
  - label\_final\_steps, 46
  - label\_generations, 46
  - label\_iterations, 46
  - label\_max, 46
  - label\_min, 47
  - label\_mutation, 47
  - label\_p, 47

- label\_population, [47](#)
- label\_precision, [47](#)
- label\_relaxation, [47](#)
- label\_reproduction, [48](#)
- label\_result, [48](#)
- label\_simulations, [48](#)
- label\_simulator, [48](#)
- label\_step, [48](#)
- label\_steps, [48](#)
- label\_sweeps, [49](#)
- label\_threshold, [49](#)
- label\_tolerance, [49](#)
- label\_variable, [49](#)
- label\_variables, [49](#)
- label\_weight, [49](#)
- logo, [50](#)
- nexperiments, [50](#)
- nvariables, [50](#)
- scrolled\_max, [50](#)
- scrolled\_maxabs, [50](#)
- scrolled\_min, [50](#)
- scrolled\_minabs, [51](#)
- scrolled\_p, [51](#)
- scrolled\_step, [51](#)
- scrolled\_threshold, [51](#)
- spin\_adaptation, [51](#)
- spin\_best, [51](#)
- spin\_bits, [52](#)
- spin\_estimates, [52](#)
- spin\_final\_steps, [52](#)
- spin\_generations, [52](#)
- spin\_iterations, [52](#)
- spin\_max, [52](#)
- spin\_maxabs, [53](#)
- spin\_min, [53](#)
- spin\_minabs, [53](#)
- spin\_mutation, [53](#)
- spin\_p, [53](#)
- spin\_population, [53](#)
- spin\_precision, [54](#)
- spin\_relaxation, [54](#)
- spin\_reproduction, [54](#)
- spin\_simulations, [54](#)
- spin\_step, [54](#)
- spin\_steps, [54](#)
- spin\_sweeps, [55](#)
- spin\_threshold, [55](#)
- spin\_tolerance, [55](#)
- spin\_weight, [55](#)
- variable, [55](#)
- window, [55](#)
- window
  - interface.c, [186](#)
  - interface.h, [233](#)
  - Window, [55](#)
- window\_about
  - interface.c, [152](#)
- window\_add\_experiment
  - interface.c, [152](#)
- window\_add\_variable
  - interface.c, [153](#)
- window\_get\_algorithm
  - interface.c, [154](#)
- window\_get\_climbing
  - interface.c, [154](#)
- window\_get\_norm
  - interface.c, [155](#)
- window\_help
  - interface.c, [156](#)
- window\_inputs\_experiment
  - interface.c, [156](#)
- window\_label\_variable
  - interface.c, [157](#)
- window\_name\_experiment
  - interface.c, [157](#)
- window\_new
  - interface.c, [158](#)
  - interface.h, [222](#)
- window\_open
  - interface.c, [168](#)
- window\_precision\_variable
  - interface.c, [169](#)
- window\_rangemax\_variable
  - interface.c, [170](#)
- window\_rangemaxabs\_variable
  - interface.c, [170](#)
- window\_rangemin\_variable
  - interface.c, [170](#)
- window\_rangeminabs\_variable
  - interface.c, [170](#)
- window\_read
  - interface.c, [171](#)
- window\_remove\_experiment
  - interface.c, [173](#)
- window\_remove\_variable
  - interface.c, [174](#)
- window\_run
  - interface.c, [174](#)
- window\_save
  - interface.c, [175](#)
- window\_save\_climbing
  - interface.c, [176](#)
- window\_set\_algorithm
  - interface.c, [177](#)
- window\_set\_experiment
  - interface.c, [178](#)
- window\_set\_variable
  - interface.c, [178](#)
- window\_step\_variable
  - interface.c, [179](#)
- window\_template\_experiment
  - interface.c, [180](#)
- window\_template\_experiment\_close
  - interface.c, [180](#)
- window\_update
  - interface.c, [181](#)

window\_update\_climbing  
    interface.c, [183](#)  
window\_update\_variable  
    interface.c, [184](#)  
window\_weight\_experiment  
    interface.c, [185](#)