MPCOTool

Generated by Doxygen 1.9.4

1 Data Structure Index 1

1.1 Data Structures . . . 1

2 File Index 3

2.1 File List . . . 3

3 Data Structure Documentation 5

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 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 climbing . . . 8

3.2.2.4 directory . . . 9

3.2.2.5 evaluator . . . 9

3.2.2.6 experiment . . . 9

3.2.2.7 mutation_ratio . . . 9

3.2.2.8 name . . . 9

3.2.2.9 nbest . . . 10

3.2.2.10 nestimates . . . 10

3.2.2.11 nexperiments . . . 10

3.2.2.12 niterations . . . 10

3.2.2.13 norm . . . 10

3.2.2.14 nsimulations . . . 11

3.2.2.15 nsteps . . . 11

3.2.2.16 nvariables . . . 11

3.2.2.17 p . . . 11

3.2.2.18 relaxation . . . 11

3.2.2.19 reproduction_ratio . . . 12

3.2.2.20 result . . . 12

3.2.2.21 seed . . . 12

3.2.2.22 simulator . . . 12

3.2.2.23 threshold . . . 12

3.2.2.24 tolerance . . . 13

3.2.2.25 type . . . 13

3.2.2.26 variable . . . 13

3.2.2.27 variables . . . 13
3.3 Optimize Struct Reference . . . 14
3.3.1 Detailed Description . . . 16
3.3.2 Field Documentation . . . 16
3.3.2.1 adaptation_ratio . . . 16
3.3.2.2 algorithm . . . 17
3.3.2.3 calculation_time . . . 17
3.3.2.4 climbing . . . 17
3.3.2.5 error_best . . . 17
3.3.2.6 error_old . . . 17
3.3.2.7 evaluator . . . 18
3.3.2.8 experiment . . . 18
3.3.2.9 file . . . 18
3.3.2.10 file_result . . . 18
3.3.2.11 file_variables . . . 18
3.3.2.12 genetic_variable . . . 19
3.3.2.13 label . . . 19
3.3.2.14 mpi_rank . . . 19
3.3.2.15 mutation_ratio . . . 19
3.3.2.16 nbest . . . 19
3.3.2.17 nbits . . . 20
3.3.2.18 nend . . . 20
3.3.2.19 nend_climbing . . . 20
3.3.2.20 nestimates . . . 20
3.3.2.21 nexperiments . . . 20
3.3.2.22 ninputs . . . 21
3.3.2.23 niterations . . . 21
3.3.2.24 nsaveds . . . 21
3.3.2.25 nsimulations . . . 21
3.3.2.26 nstart . . . 21
3.3.2.27 nstart_climbing . . . 22
3.3.2.28 nsteps . . . 22
3.3.2.29 nsweeps . . . 22
3.3.2.30 nvariables . . . 22
3.3.2.31 p . . . 22
3.3.2.32 precision . . . 23
3.3.2.33 rangemax . . . 23
3.3.2.34 rangemaxabs . . . 23
3.3.2.35 rangemin . . . 23
3.3.2.36 rangeminabs . . . 23
3.3.2.37 relaxation . . . 24
3.3.2.38 reproduction_ratio . . . 24

3.3.2.39 result . . . 24
3.3.2.40 rng . . . 24
3.3.2.41 seed . . . 24
3.3.2.42 simulation_best . . . 25
3.3.2.43 simulator . . . 25
3.3.2.44 step . . . 25
3.3.2.45 stop . . . 25
3.3.2.46 thread . . . 25
3.3.2.47 thread_climbing . . . 26
3.3.2.48 threshold . . . 26
3.3.2.49 tolerance . . . 26
3.3.2.50 value . . . 26
3.3.2.51 value_old . . . 26
3.3.2.52 variables . . . 27
3.3.2.53 weight . . . 27
3.4 Options Struct Reference . . . 27
3.4.1 Detailed Description . . . 28
3.4.2 Field Documentation . . . 28
3.4.2.1 dialog . . . 28
3.4.2.2 grid . . . 28
3.4.2.3 label_climbing . . . 28
3.4.2.4 label_seed . . . 28
3.4.2.5 label_threads . . . 29
3.4.2.6 spin_climbing . . . 29
3.4.2.7 spin_seed . . . 29
3.4.2.8 spin_threads . . . 29
3.5 ParallelData Struct Reference . . . 29
3.5.1 Detailed Description . . . 30
3.5.2 Field Documentation . . . 30
3.5.2.1 thread . . . 30
3.6 Running Struct Reference . . . 30
3.6.1 Detailed Description . . . 31
3.6.2 Field Documentation . . . 31
3.6.2.1 dialog . . . 31
3.6.2.2 grid . . . 31
3.6.2.3 label . . . 31
3.6.2.4 spinner . . . 31
3.7 Variable Struct Reference . . . 32
3.7.1 Detailed Description . . . 32
3.7.2 Field Documentation . . . 32
3.7.2.1 name . . . 32
3.7.2.2 nbits . . . 33

3.7.2.3 nsweeps . . . 33
3.7.2.4 precision . . . 33
3.7.2.5 rangemax . . . 33
3.7.2.6 rangemaxabs . . . 33
3.7.2.7 rangemin . . . 34
3.7.2.8 rangeminabs . . . 34
3.7.2.9 step . . . 34
3.8 Window Struct Reference . . . 34
3.8.1 Detailed Description . . . 39
3.8.2 Field Documentation . . . 39
3.8.2.1 application_directory . . . 39
3.8.2.2 box_buttons . . . 39
3.8.2.3 button_about . . . 40
3.8.2.4 button_add_experiment . . . 40
3.8.2.5 button_add_variable . . . 40
3.8.2.6 button_algorithm . . . 40
3.8.2.7 button_climbing . . . 40
3.8.2.8 button_evaluator . . . 41
3.8.2.9 button_exit . . . 41
3.8.2.10 button_experiment . . . 41
3.8.2.11 button_help . . . 41
3.8.2.12 button_norm . . . 41
3.8.2.13 button_open . . . 42
3.8.2.14 button_options . . . 42
3.8.2.15 button_remove_experiment . . . 42
3.8.2.16 button_remove_variable . . . 42
3.8.2.17 button_run . . . 42
3.8.2.18 button_save . . . 43
3.8.2.19 button_simulator . . . 43
3.8.2.20 button_template . . . 43
3.8.2.21 check_climbing . . . 43
3.8.2.22 check_evaluator . . . 43
3.8.2.23 check_maxabs . . . 44
3.8.2.24 check_minabs . . . 44
3.8.2.25 check_template . . . 44
3.8.2.26 combo_experiment . . . 44
3.8.2.27 combo_variable . . . 44
3.8.2.28 entry_result . . . 45
3.8.2.29 entry_variable . . . 45
3.8.2.30 entry_variables . . . 45
3.8.2.31 experiment . . . 45
3.8.2.32 frame_algorithm . . . 45

3.8.2.33 frame_experiment . . . 46
3.8.2.34 frame_norm . . . 46
3.8.2.35 frame_variable . . . 46
3.8.2.36 grid . . . 46
3.8.2.37 grid_algorithm . . . 46
3.8.2.38 grid_climbing . . . 47
3.8.2.39 grid_experiment . . . 47
3.8.2.40 grid_files . . . 47
3.8.2.41 grid_norm . . . 47
3.8.2.42 grid_variable . . . 47
3.8.2.43 id_experiment . . . 48
3.8.2.44 id_experiment_name . . . 48
3.8.2.45 id_input . . . 48
3.8.2.46 id_template . . . 48
3.8.2.47 id_variable . . . 48
3.8.2.48 id_variable_label . . . 49
3.8.2.49 label_adaptation . . . 49
3.8.2.50 label_bests . . . 49
3.8.2.51 label_bits . . . 49
3.8.2.52 label_estimates . . . 49
3.8.2.53 label_experiment . . . 50
3.8.2.54 label_generations . . . 50
3.8.2.55 label_iterations . . . 50
3.8.2.56 label_max . . . 50
3.8.2.57 label_min . . . 50
3.8.2.58 label_mutation . . . 51
3.8.2.59 label_p . . . 51
3.8.2.60 label_population . . . 51
3.8.2.61 label_precision . . . 51
3.8.2.62 label_relaxation . . . 51
3.8.2.63 label_reproduction . . . 52
3.8.2.64 label_result . . . 52
3.8.2.65 label_simulations . . . 52
3.8.2.66 label_simulator . . . 52
3.8.2.67 label_step . . . 52
3.8.2.68 label_steps . . . 53
3.8.2.69 label_sweeps . . . 53
3.8.2.70 label_threshold . . . 53
3.8.2.71 label_tolerance . . . 53
3.8.2.72 label_variable . . . 53
3.8.2.73 label_variables . . . 54
3.8.2.74 label_weight . . . 54

3.8.2.75 logo . . . 54
3.8.2.76 nexperiments . . . 54
3.8.2.77 nvariables . . . 54
3.8.2.78 scrolled_max . . . 55
3.8.2.79 scrolled_maxabs . . . 55
3.8.2.80 scrolled_min . . . 55
3.8.2.81 scrolled_minabs . . . 55
3.8.2.82 scrolled_p . . . 55
3.8.2.83 scrolled_step . . . 56
3.8.2.84 scrolled_threshold . . . 56
3.8.2.85 spin_adaptation . . . 56
3.8.2.86 spin_bests . . . 56
3.8.2.87 spin_bits . . . 56
3.8.2.88 spin_estimates . . . 57
3.8.2.89 spin_generations . . . 57
3.8.2.90 spin_iterations . . . 57
3.8.2.91 spin_max . . . 57
3.8.2.92 spin_maxabs . . . 57
3.8.2.93 spin_min . . . 58
3.8.2.94 spin_minabs . . . 58
3.8.2.95 spin_mutation . . . 58
3.8.2.96 spin_p . . . 58
3.8.2.97 spin_population . . . 58
3.8.2.98 spin_precision . . . 59
3.8.2.99 spin_relaxation . . . 59
3.8.2.100 spin_reproduction . . . 59
3.8.2.101 spin_simulations . . . 59
3.8.2.102 spin_step . . . 59
3.8.2.103 spin_steps . . . 60
3.8.2.104 spin_sweeps . . . 60
3.8.2.105 spin_threshold . . . 60
3.8.2.106 spin_tolerance . . . 60
3.8.2.107 spin_weight . . . 60
3.8.2.108 variable . . . 61
3.8.2.109 window . . . 61

4 File Documentation **63**

4.1 config.h File Reference . . . 63
4.1.1 Detailed Description . . . 66
4.1.2 Macro Definition Documentation . . . 66
4.1.2.1 DEFAULT_PRECISION . . . 66
4.1.2.2 DEFAULT_RANDOM_SEED . . . 66

4.1.2.3 DEFAULT_RELAXATION . . . 67
4.1.2.4 HAVE_MPI . . . 67
4.1.2.5 LABEL_ABSOLUTE_MAXIMUM . . . 67
4.1.2.6 LABEL_ABSOLUTE_MINIMUM . . . 67
4.1.2.7 LABEL_ADAPTATION . . . 67
4.1.2.8 LABEL_ALGORITHM . . . 68
4.1.2.9 LABEL_CLIMBING . . . 68
4.1.2.10 LABEL_COORDINATES . . . 68
4.1.2.11 LABEL_EUCLIDIAN . . . 68
4.1.2.12 LABEL_EVALUATOR . . . 68
4.1.2.13 LABEL_EXPERIMENT . . . 69
4.1.2.14 LABEL_EXPERIMENTS . . . 69
4.1.2.15 LABEL_GENETIC . . . 69
4.1.2.16 LABEL_MAXIMUM . . . 69
4.1.2.17 LABEL_MINIMUM . . . 69
4.1.2.18 LABEL_MONTE_CARLO . . . 70
4.1.2.19 LABEL_MUTATION . . . 70
4.1.2.20 LABEL_NAME . . . 70
4.1.2.21 LABEL_NBEST . . . 70
4.1.2.22 LABEL_NBITS . . . 70
4.1.2.23 LABEL_NESTIMATES . . . 71
4.1.2.24 LABEL_NGENERATIONS . . . 71
4.1.2.25 LABEL_NITERATIONS . . . 71
4.1.2.26 LABEL_NORM . . . 71
4.1.2.27 LABEL_NPOPULATION . . . 71
4.1.2.28 LABEL_NSIMULATIONS . . . 72
4.1.2.29 LABEL_NSTEPS . . . 72
4.1.2.30 LABEL_NSWEEPS . . . 72
4.1.2.31 LABEL_OPTIMIZE . . . 72
4.1.2.32 LABEL_ORTHOGONAL . . . 72
4.1.2.33 LABEL_P . . . 73
4.1.2.34 LABEL_PRECISION . . . 73
4.1.2.35 LABEL_RANDOM . . . 73
4.1.2.36 LABEL_RELAXATION . . . 73
4.1.2.37 LABEL_REPRODUCTION . . . 73
4.1.2.38 LABEL_RESULT_FILE . . . 74
4.1.2.39 LABEL_SEED . . . 74
4.1.2.40 LABEL_SIMULATOR . . . 74
4.1.2.41 LABEL_STEP . . . 74
4.1.2.42 LABEL_SWEEP . . . 74
4.1.2.43 LABEL_TAXICAB . . . 75
4.1.2.44 LABEL_TEMPLATE1 . . . 75

4.1.2.45 LABEL_TEMPLATE2 . . . 75
4.1.2.46 LABEL_TEMPLATE3 . . . 75
4.1.2.47 LABEL_TEMPLATE4 . . . 75
4.1.2.48 LABEL_TEMPLATE5 . . . 76
4.1.2.49 LABEL_TEMPLATE6 . . . 76
4.1.2.50 LABEL_TEMPLATE7 . . . 76
4.1.2.51 LABEL_TEMPLATE8 . . . 76
4.1.2.52 LABEL_THRESHOLD . . . 76
4.1.2.53 LABEL_TOLERANCE . . . 77
4.1.2.54 LABEL_VARIABLE . . . 77
4.1.2.55 LABEL_VARIABLES . . . 77
4.1.2.56 LABEL_VARIABLES_FILE . . . 77
4.1.2.57 LABEL_WEIGHT . . . 77
4.1.2.58 LOCALE_DIR . . . 78
4.1.2.59 MAX_NINPUTS . . . 78
4.1.2.60 NALGORITHMS . . . 78
4.1.2.61 NCLIMBINGS . . . 78
4.1.2.62 NNORMS . . . 78
4.1.2.63 NPRECISIONS . . . 79
4.1.2.64 PROGRAM_INTERFACE . . . 79
4.1.3 Enumeration Type Documentation . . . 79
4.1.3.1 INPUT_TYPE . . . 79
4.2 config.h . . . 79
4.3 experiment.c File Reference . . . 81
4.3.1 Detailed Description . . . 82
4.3.2 Macro Definition Documentation . . . 82
4.3.2.1 DEBUG_EXPERIMENT . . . 82
4.3.3 Function Documentation . . . 82
4.3.3.1 experiment_error() . . . 82
4.3.3.2 experiment_free() . . . 83
4.3.3.3 experiment_new() . . . 83
4.3.3.4 experiment_open_json() . . . 84
4.3.3.5 experiment_open_xml() . . . 86
4.3.4 Variable Documentation . . . 88
4.3.4.1 stencil . . . 88
4.4 experiment.c . . . 88
4.5 experiment.h File Reference . . . 92
4.5.1 Detailed Description . . . 93
4.5.2 Function Documentation . . . 93
4.5.2.1 experiment_error() . . . 93
4.5.2.2 experiment_free() . . . 93
4.5.2.3 experiment_open_json() . . . 94

4.5.2.4 experiment_open_xml() . . . 96
4.5.3 Variable Documentation . . . 98
4.5.3.1 stencil . . . 98
4.6 experiment.h . . . 98
4.7 input.c File Reference . . . 99
4.7.1 Detailed Description . . . 100
4.7.2 Macro Definition Documentation . . . 100
4.7.2.1 DEBUG_INPUT . . . 100
4.7.3 Function Documentation . . . 100
4.7.3.1 input_error() . . . 100
4.7.3.2 input_free() . . . 101
4.7.3.3 input_new() . . . 102
4.7.3.4 input_open() . . . 102
4.7.3.5 input_open_json() . . . 103
4.7.3.6 input_open_xml() . . . 109
4.7.4 Variable Documentation . . . 115
4.7.4.1 input . . . 115
4.7.4.2 result_name . . . 115
4.7.4.3 variables_name . . . 116
4.8 input.c . . . 116
4.9 input.h File Reference . . . 127
4.9.1 Detailed Description . . . 128
4.9.2 Enumeration Type Documentation . . . 128
4.9.2.1 ClimbingMethod . . . 128
4.9.2.2 ErrorNorm . . . 129
4.9.3 Function Documentation . . . 129
4.9.3.1 input_free() . . . 129
4.9.3.2 input_new() . . . 130
4.9.3.3 input_open() . . . 131
4.9.4 Variable Documentation . . . 132
4.9.4.1 input . . . 132
4.9.4.2 result_name . . . 132
4.9.4.3 variables_name . . . 133
4.10 input.h . . . 133
4.11 interface.c File Reference . . . 134
4.11.1 Detailed Description . . . 136
4.11.2 Macro Definition Documentation . . . 136
4.11.2.1 DEBUG_INTERFACE . . . 136
4.11.2.2 INPUT_FILE . . . 136
4.11.3 Function Documentation . . . 137
4.11.3.1 dialog_evaluator() . . . 137
4.11.3.2 dialog_evaluator_close() . . . 137

4.11.3.3 dialog_name_experiment_close() . . . 138
4.11.3.4 dialog_open_close() . . . 139
4.11.3.5 dialog_options_close() . . . 140
4.11.3.6 dialog_save_close() . . . 141
4.11.3.7 dialog_simulator() . . . 142
4.11.3.8 dialog_simulator_close() . . . 143
4.11.3.9 input_save() . . . 144
4.11.3.10 input_save_climbing_json() . . . 145
4.11.3.11 input_save_climbing_xml() . . . 145
4.11.3.12 input_save_json() . . . 146
4.11.3.13 input_save_xml() . . . 149
4.11.3.14 options_new() . . . 151
4.11.3.15 running_new() . . . 152
4.11.3.16 window_about() . . . 153
4.11.3.17 window_add_experiment() . . . 153
4.11.3.18 window_add_variable() . . . 154
4.11.3.19 window_get_algorithm() . . . 155
4.11.3.20 window_get_climbing() . . . 156
4.11.3.21 window_get_norm() . . . 156
4.11.3.22 window_help() . . . 157
4.11.3.23 window_inputs_experiment() . . . 157
4.11.3.24 window_label_variable() . . . 158
4.11.3.25 window_name_experiment() . . . 159
4.11.3.26 window_new() . . . 159
4.11.3.27 window_open() . . . 169
4.11.3.28 window_precision_variable() . . . 170
4.11.3.29 window_rangemax_variable() . . . 171
4.11.3.30 window_rangemaxabs_variable() . . . 171
4.11.3.31 window_rangemin_variable() . . . 171
4.11.3.32 window_rangeminabs_variable() . . . 172
4.11.3.33 window_read() . . . 172
4.11.3.34 window_remove_experiment() . . . 174
4.11.3.35 window_remove_variable() . . . 175
4.11.3.36 window_run() . . . 176
4.11.3.37 window_save() . . . 177
4.11.3.38 window_save_climbing() . . . 178
4.11.3.39 window_set_algorithm() . . . 178
4.11.3.40 window_set_experiment() . . . 179
4.11.3.41 window_set_variable() . . . 179
4.11.3.42 window_step_variable() . . . 181
4.11.3.43 window_template_experiment() . . . 181
4.11.3.44 window_template_experiment_close() . . . 182

4.11.3.45 window_update() . . . 183
4.11.3.46 window_update_climbing() . . . 185
4.11.3.47 window_update_variable() . . . 185
4.11.3.48 window_weight_experiment() . . . 186
4.11.4 Variable Documentation . . . 186
4.11.4.1 logo . . . 186
4.11.4.2 options . . . 187
4.11.4.3 running . . . 187
4.11.4.4 window . . . 187
4.12 interface.c . . . 187
4.13 interface.h File Reference . . . 220
4.13.1 Detailed Description . . . 221
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 . . . 232
4.13.4.1 window . . . 232
4.14 interface.h . . . 232
4.15 main.c File Reference . . . 235
4.15.1 Detailed Description . . . 235
4.15.2 Function Documentation . . . 236
4.15.2.1 main() . . . 236
4.16 main.c . . . 236
4.17 mpcotool.c File Reference . . . 237
4.17.1 Detailed Description . . . 238
4.17.2 Macro Definition Documentation . . . 238
4.17.2.1 DEBUG_MPCOTOOL . . . 238
4.17.3 Function Documentation . . . 239
4.17.3.1 mpcotool() . . . 239
4.18 mpcotool.c . . . 241
4.19 mpcotool.h File Reference . . . 243
4.19.1 Detailed Description . . . 244
4.19.2 Function Documentation . . . 244
4.19.2.1 mpcotool() . . . 244
4.20 mpcotool.h . . . 246
4.21 optimize.c File Reference . . . 246
4.21.1 Detailed Description . . . 248
4.21.2 Macro Definition Documentation . . . 248
4.21.2.1 DEBUG_OPTIMIZE . . . 249
4.21.2.2 RM . . . 249
4.21.3 Function Documentation . . . 249

4.21.3.1 optimize_best() . . . 249
4.21.3.2 optimize_best_climbing() . . . 250
4.21.3.3 optimize_climbing() . . . 250
4.21.3.4 optimize_climbing_sequential() . . . 251
4.21.3.5 optimize_climbing_thread() . . . 252
4.21.3.6 optimize_estimate_climbing_coordinates() . . . 253
4.21.3.7 optimize_estimate_climbing_random() . . . 254
4.21.3.8 optimize_free() . . . 254
4.21.3.9 optimize_genetic() . . . 255
4.21.3.10 optimize_genetic_objective() . . . 255
4.21.3.11 optimize_input() . . . 256
4.21.3.12 optimize_iterate() . . . 257
4.21.3.13 optimize_merge() . . . 258
4.21.3.14 optimize_merge_old() . . . 259
4.21.3.15 optimize_MonteCarlo() . . . 260
4.21.3.16 optimize_norm_euclidian() . . . 260
4.21.3.17 optimize_norm_maximum() . . . 261
4.21.3.18 optimize_norm_p() . . . 262
4.21.3.19 optimize_norm_taxicab() . . . 263
4.21.3.20 optimize_open() . . . 264
4.21.3.21 optimize_orthogonal() . . . 268
4.21.3.22 optimize_parse() . . . 268
4.21.3.23 optimize_print() . . . 270
4.21.3.24 optimize_refine() . . . 271
4.21.3.25 optimize_save_old() . . . 272
4.21.3.26 optimize_save_variables() . . . 272
4.21.3.27 optimize_sequential() . . . 273
4.21.3.28 optimize_step() . . . 274
4.21.3.29 optimize_step_climbing() . . . 274
4.21.3.30 optimize_sweep() . . . 275
4.21.3.31 optimize_synchronise() . . . 276
4.21.3.32 optimize_thread() . . . 277
4.21.4 Variable Documentation . . . 277
4.21.4.1 nthreads_climbing . . . 278
4.21.4.2 optimize . . . 278
4.21.4.3 optimize_algorithm . . . 278
4.21.4.4 optimize_estimate_climbing . . . 278
4.21.4.5 optimize_norm . . . 278
4.22 optimize.c . . . 279
4.23 optimize.h File Reference . . . 297
4.23.1 Detailed Description . . . 298
4.23.2 Function Documentation . . . 298

4.23.2.1 optimize_free() . . . 298
4.23.2.2 optimize_open() . . . 299
4.23.3 Variable Documentation . . . 303
4.23.3.1 nthreads_climbing . . . 303
4.23.3.2 optimize . . . 303
4.24 optimize.h . . . 303
4.25 tools.c File Reference . . . 304
4.25.1 Detailed Description . . . 305
4.25.2 Function Documentation . . . 305
4.25.2.1 gtk_array_get_active() . . . 305
4.25.2.2 process_pending() . . . 306
4.25.3 Variable Documentation . . . 306
4.25.3.1 error_message . . . 306
4.25.3.2 main_window . . . 306
4.25.3.3 show_pending . . . 307
4.26 tools.c . . . 307
4.27 tools.h File Reference . . . 308
4.27.1 Detailed Description . . . 309
4.27.2 Macro Definition Documentation . . . 309
4.27.2.1 ERROR_TYPE . . . 309
4.27.2.2 G_APPLICATION_DEFAULT_FLAGS . . . 309
4.27.2.3 INFO_TYPE . . . 309
4.27.3 Function Documentation . . . 309
4.27.3.1 gtk_array_get_active() . . . 309
4.27.3.2 process_pending() . . . 310
4.27.4 Variable Documentation . . . 310
4.27.4.1 error_message . . . 310
4.27.4.2 main_window . . . 310
4.27.4.3 show_pending . . . 311
4.28 tools.h . . . 311
4.29 variable.c File Reference . . . 312
4.29.1 Detailed Description . . . 312
4.29.2 Macro Definition Documentation . . . 313
4.29.2.1 DEBUG_VARIABLE . . . 313
4.29.3 Function Documentation . . . 313
4.29.3.1 variable_error() . . . 313
4.29.3.2 variable_free() . . . 313
4.29.3.3 variable_open_json() . . . 314
4.29.3.4 variable_open_xml() . . . 317
4.29.4 Variable Documentation . . . 319
4.29.4.1 format . . . 319
4.29.4.2 precision . . . 320

4.30 variable.c . . . 320
4.31 variable.h File Reference . . . 324
4.31.1 Detailed Description . . . 325
4.31.2 Enumeration Type Documentation . . . 326
4.31.2.1 Algorithm . . . 326
4.31.3 Function Documentation . . . 326
4.31.3.1 variable_error() . . . 326
4.31.3.2 variable_free() . . . 327
4.31.3.3 variable_open_json() . . . 327
4.31.3.4 variable_open_xml() . . . 330
4.31.4 Variable Documentation . . . 333
4.31.4.1 format . . . 333
4.31.4.2 precision . . . 333
4.32 variable.h . . . 334

Index **335**

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Experiment
: Struct to define the experiment data 5

Input
: Struct to define the optimization input file 6

Optimize
: Struct to define the optimization ation data 14

Options
: Struct to define the options dialog 27

ParallelData
: Struct to pass to the GThreads parallelized function 29

Running
: Struct to define the running dialog 30

Variable
: Struct to define the variable data 32

Window
: Struct to define the main window 34

Chapter 2

File Index

2.1 File List

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

config.h
Configuration header file . . . 63
experiment.c
Source file to define the experiment data . . . 81
experiment.h
Header file to define the experiment data . . . 92
input.c
Source file to define the input functions . . . 99
input.h
Header file to define the input functions . . . 127
interface.c
Source file to define the graphical interface functions . . . 134
interface.h
Header file to define the graphical interface functions . . . 220
main.c
Main source file . . . 235
mpcotool.c
Main function source file . . . 237
mpcotool.h
Main function header file . . . 243
optimize.c
Source file to define the optimization functions . . . 246
optimize.h
Header file to define the optimization functions . . . 297
tools.c
Source file to define some useful functions . . . 304
tools.h
Header file to define some useful functions . . . 308
variable.c
Source file to define the variable data . . . 312
variable.h
Header file to define the variable data . . . 324

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.

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 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:

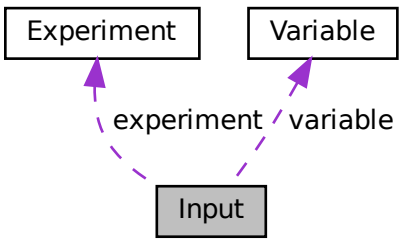
- experiment.h

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 * 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 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.

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 79 of file input.h.

3.2.2.2 algorithm

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

Algorithm type.

Definition at line 88 of file input.h.

3.2.2.3 climbing

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

Method to estimate the hill climbing.

Definition at line 91 of file input.h.

3.2.2.4 directory

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

Working directory.

Definition at line 74 of file input.h.

3.2.2.5 evaluator

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

Name of the program to evaluate the objective function.

Definition at line 72 of file input.h.

3.2.2.6 experiment

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

Array or experiments.

Definition at line 67 of file input.h.

3.2.2.7 mutation_ratio

```
double Input::mutation_ratio
```

Mutation probability.

Definition at line 77 of file input.h.

3.2.2.8 name

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

Input data file name.

Definition at line 75 of file input.h.

3.2.2.9 nbest

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

Number of best simulations.

Definition at line 95 of file input.h.

3.2.2.10 nestimates

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

Number of simulations to estimate the hill climbing.

Definition at line 92 of file input.h.

3.2.2.11 nexperiments

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

Experiments number.

Definition at line 86 of file input.h.

3.2.2.12 niterations

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

Number of algorithm iterations.

Definition at line 94 of file input.h.

3.2.2.13 norm

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

Error norm type.

Definition at line 96 of file input.h.

3.2.2.14 nsimulations

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

Simulations number per experiment.

Definition at line 87 of file input.h.

3.2.2.15 nsteps

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

Number of steps to do the hill climbing method.

Definition at line 89 of file input.h.

3.2.2.16 nvariables

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

Variables number.

Definition at line 85 of file input.h.

3.2.2.17 p

```
double Input::p
```

Exponent of the P error norm.

Definition at line 81 of file input.h.

3.2.2.18 relaxation

```
double Input::relaxation
```

Relaxation parameter.

Definition at line 80 of file input.h.

3.2.2.19 reproduction_ratio

```
double Input::reproduction_ratio
```

Reproduction probability.

Definition at line 78 of file input.h.

3.2.2.20 result

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

Name of the result file.

Definition at line 69 of file input.h.

3.2.2.21 seed

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

Seed of the pseudo-random numbers generator.

Definition at line 83 of file input.h.

3.2.2.22 simulator

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

Name of the simulator program.

Definition at line 71 of file input.h.

3.2.2.23 threshold

```
double Input::threshold
```

Threshold to finish the optimization.

Definition at line 82 of file input.h.

3.2.2.24 tolerance

```
double Input::tolerance
```

Algorithm tolerance.

Definition at line 76 of file input.h.

3.2.2.25 type

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

Type of input file.

Definition at line 97 of file input.h.

3.2.2.26 variable

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

Array of variables.

Definition at line 68 of file input.h.

3.2.2.27 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:

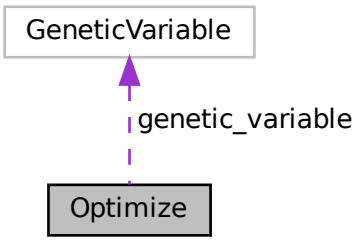
- input.h

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.
- 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 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.
- 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 86 of file optimize.h.

3.3.2.2 algorithm

```
unsigned int Optimize::algorithm
```

Algorithm type.

Definition at line 101 of file optimize.h.

3.3.2.3 calculation_time

```
double Optimize::calculation_time
```

Calculation time.

Definition at line 88 of file optimize.h.

3.3.2.4 climbing

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

Vector of hill climbing estimation.

Definition at line 68 of file optimize.h.

3.3.2.5 error_best

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

Array of the best minimum errors.

Definition at line 65 of file optimize.h.

3.3.2.6 error_old

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

Array of the best minimum errors on the previous step.

Definition at line 71 of file optimize.h.

3.3.2.7 evaluator

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

Name of the program to evaluate the objective function.

Definition at line 58 of file optimize.h.

3.3.2.8 experiment

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

Array of experimental data file names.

Definition at line 48 of file optimize.h.

3.3.2.9 file

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

Matrix of input template files.

Definition at line 47 of file optimize.h.

3.3.2.10 file_result

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

Result file.

Definition at line 53 of file optimize.h.

3.3.2.11 file_variables

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

Variables file.

Definition at line 54 of file optimize.h.

3.3.2.12 genetic_variable

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

Array of variables for the genetic algorithm.

Definition at line 51 of file optimize.h.

3.3.2.13 label

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

Array of variable names.

Definition at line 49 of file optimize.h.

3.3.2.14 mpi_rank

```
int Optimize::mpi_rank
```

Number of MPI task.

Definition at line 113 of file optimize.h.

3.3.2.15 mutation_ratio

```
double Optimize::mutation_ratio
```

Mutation probability.

Definition at line 84 of file optimize.h.

3.3.2.16 nbest

```
unsigned int Optimize::nbest
```

Number of best simulations.

Definition at line 109 of file optimize.h.

3.3.2.17 nbits

```
unsigned int* Optimize::nbits
```

Array of bits number of the genetic algorithm.

Definition at line 75 of file optimize.h.

3.3.2.18 nend

```
unsigned int Optimize::nend
```

Ending simulation number of the task.

Definition at line 103 of file optimize.h.

3.3.2.19 nend_climbing

```
unsigned int Optimize::nend_climbing
```

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

Definition at line 106 of file optimize.h.

3.3.2.20 nestimates

```
unsigned int Optimize::nestimates
```

Number of simulations to estimate the climbing.

Definition at line 99 of file optimize.h.

3.3.2.21 nexperiments

```
unsigned int Optimize::nexperiments
```

Experiments number.

Definition at line 94 of file optimize.h.

3.3.2.22 ninputs

```
unsigned int Optimize::ninputs
```

Number of input files to the simulator.

Definition at line 95 of file optimize.h.

3.3.2.23 niterations

```
unsigned int Optimize::niterations
```

Number of algorithm iterations.

Definition at line 108 of file optimize.h.

3.3.2.24 nsaveds

```
unsigned int Optimize::nsaveds
```

Number of saved simulations.

Definition at line 110 of file optimize.h.

3.3.2.25 nsimulations

```
unsigned int Optimize::nsimulations
```

Simulations number per experiment.

Definition at line 96 of file optimize.h.

3.3.2.26 nstart

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

Beginning simulation number of the task.

Definition at line 102 of file optimize.h.

3.3.2.27 nstart_climbing

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

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

Definition at line 104 of file optimize.h.

3.3.2.28 nsteps

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

Number of steps for the hill climbing method.

Definition at line 97 of file optimize.h.

3.3.2.29 nsweeps

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

Array of sweeps of the sweep algorithm.

Definition at line 74 of file optimize.h.

3.3.2.30 nvariables

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

Variables number.

Definition at line 93 of file optimize.h.

3.3.2.31 p

```
double Optimize::p
```

Exponent of the P error norm.

Definition at line 89 of file optimize.h.

3.3.2.32 precision

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

Array of variable precisions.

Definition at line 73 of file optimize.h.

3.3.2.33 rangemax

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

Array of maximum variable values.

Definition at line 62 of file optimize.h.

3.3.2.34 rangemaxabs

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

Array of absolute maximum variable values.

Definition at line 64 of file optimize.h.

3.3.2.35 rangemin

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

Array of minimum variable values.

Definition at line 61 of file optimize.h.

3.3.2.36 rangeminabs

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

Array of absolute minimum variable values.

Definition at line 63 of file optimize.h.

3.3.2.37 relaxation

```
double Optimize::relaxation
```

Relaxation parameter.

Definition at line 87 of file optimize.h.

3.3.2.38 reproduction_ratio

```
double Optimize::reproduction_ratio
```

Reproduction probability.

Definition at line 85 of file optimize.h.

3.3.2.39 result

```
char* Optimize::result
```

Name of the result file.

Definition at line 55 of file optimize.h.

3.3.2.40 rng

```
gsl_rng* Optimize::rng
```

GSL random number generator.

Definition at line 50 of file optimize.h.

3.3.2.41 seed

```
unsigned long int Optimize::seed
```

Seed of the pseudo-random numbers generator.

Definition at line 91 of file optimize.h.

3.3.2.42 simulation_best

```
unsigned int* Optimize::simulation_best
```

Array of best simulation numbers.

Definition at line 82 of file optimize.h.

3.3.2.43 simulator

```
char* Optimize::simulator
```

Name of the simulator program.

Definition at line 57 of file optimize.h.

3.3.2.44 step

```
double* Optimize::step
```

Array of hill climbing method step sizes.

Definition at line 67 of file optimize.h.

3.3.2.45 stop

```
unsigned int Optimize::stop
```

To stop the simulations.

Definition at line 111 of file optimize.h.

3.3.2.46 thread

```
unsigned int* Optimize::thread
```

Array of simulation numbers to calculate on the thread.

Definition at line 77 of file optimize.h.

3.3.2.47 thread_climbing

```
unsigned int* Optimize::thread_climbing
```

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

Definition at line 79 of file optimize.h.

3.3.2.48 threshold

```
double Optimize::threshold
```

Threshold to finish the optimization.

Definition at line 90 of file optimize.h.

3.3.2.49 tolerance

```
double Optimize::tolerance
```

Algorithm tolerance.

Definition at line 83 of file optimize.h.

3.3.2.50 value

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

Array of variable values.

Definition at line 60 of file optimize.h.

3.3.2.51 value_old

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

Array of the best variable values on the previous step.

Definition at line 69 of file optimize.h.

3.3.2.52 variables

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

Name of the variables file.

Definition at line 56 of file optimize.h.

3.3.2.53 weight

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

Array of the experiment weights.

Definition at line 66 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 121 of file optimize.h.

3.5.2 Field Documentation

3.5.2.1 thread

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

Thread number.

Definition at line 123 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

- GtkDialog * dialog
  Main GtkDialog.
- GtkLabel * label
  Label GtkLabel.
- GtkSpinner * spinner
  Animation GtkSpinner.
- GtkGrid * grid
  Grid GtkGrid.

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:

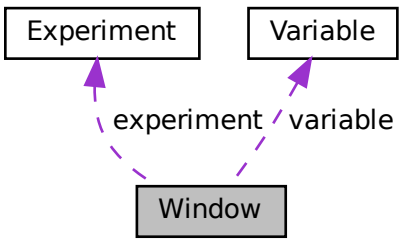
- variable.h

3.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

- GtkWindow * window
  Main GtkWindow.
- GtkGrid * grid
  Main GtkGrid.
- GtkBox * box_buttons
  GtkBox to store the main buttons.
- GtkButton * button_open
  Open GtkButton.
- GtkButton * button_save
  Save GtkButton.
- GtkButton * button_run
  Run GtkButton.
- GtkButton * button_options
  Options GtkButton.
- 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.
- 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.
- GtkCheckButton * check_climbing

  GtkCheckButton 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_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.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkLabel * label_step

  GtkLabel to set the step.
- GtkSpinButton * spin_step

  GtkSpinButton to set the step.
- GtkScrolledWindow * scrolled_step

  step GtkScrolledWindow.
- GtkFrame * frame_experiment

  Experiment GtkFrame.
- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkButton * button_experiment

  GtkButton to set the experimental data file.
- GtkLabel * label_weight

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

  Array of GtkCheckButtons to set the input templates.
- GtkButton * 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 215 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 201 of file interface.h.

3.8.2.5 button_add_variable

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

GtkButton to add a variable.

Definition at line 173 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 115 of file interface.h.

3.8.2.7 button_climbing

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

Array of GtkRadioButtons array to set the hill climbing method.

Definition at line 150 of file interface.h.

3.8.2.8 button_evaluator

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

Evaluator program GtkButton.

Definition at line 94 of file interface.h.

3.8.2.9 button_exit

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

Exit GtkButton.

Definition at line 89 of file interface.h.

3.8.2.10 button_experiment

```
GtkButton* Window::button_experiment
```

GtkButton to set the experimental data file.

Definition at line 204 of file interface.h.

3.8.2.11 button_help

```
GtkButton* Window::button_help
```

Help GtkButton.

Definition at line 87 of file interface.h.

3.8.2.12 button_norm

```
GtkRadioButton* Window::button_norm[NNORMS]
```

Array of GtkRadioButtons to set the error norm.

Definition at line 102 of file interface.h.

3.8.2.13 button_open

```
GtkButton* Window::button_open
```

Open GtkButton.

Definition at line 83 of file interface.h.

3.8.2.14 button_options

```
GtkButton* Window::button_options
```

Options GtkButton.

Definition at line 86 of file interface.h.

3.8.2.15 button_remove_experiment

```
GtkButton* Window::button_remove_experiment
```

GtkButton to remove a experiment.

Definition at line 202 of file interface.h.

3.8.2.16 button_remove_variable

```
GtkButton* Window::button_remove_variable
```

GtkButton to remove a variable.

Definition at line 174 of file interface.h.

3.8.2.17 button_run

```
GtkButton* Window::button_run
```

Run GtkButton.

Definition at line 85 of file interface.h.

3.8.2.18 button_save

```
GtkButton* Window::button_save
```

Save GtkButton.

Definition at line 84 of file interface.h.

3.8.2.19 button_simulator

```
GtkButton* Window::button_simulator
```

Simulator program GtkButton.

Definition at line 92 of file interface.h.

3.8.2.20 button_template

```
GtkButton* Window::button_template[MAX_NINPUTS]
```

Array of GtkButtons to set the input templates.

Definition at line 210 of file interface.h.

3.8.2.21 check_climbing

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

GtkCheckButton to check running the hill climbing method.

Definition at line 145 of file interface.h.

3.8.2.22 check_evaluator

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

Evaluator program GtkCheckButton.

Definition at line 93 of file interface.h.

3.8.2.23 check_maxabs

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

Absolute maximum GtkCheckButton.

Definition at line 186 of file interface.h.

3.8.2.24 check_minabs

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

Absolute minimum GtkCheckButton.

Definition at line 183 of file interface.h.

3.8.2.25 check_template

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

Array of GtkCheckButtons to set the input templates.

Definition at line 208 of file interface.h.

3.8.2.26 combo_experiment

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

Experiment GtkComboBoxEntry.

Definition at line 200 of file interface.h.

3.8.2.27 combo_variable

```
GtkComboBoxText* Window::combo_variable
```

GtkComboBoxEntry to select a variable.

Definition at line 171 of file interface.h.

3.8.2.28 entry_result

```
GtkEntry* Window::entry_result
```

Result file GtkEntry.

Definition at line 96 of file interface.h.

3.8.2.29 entry_variable

```
GtkEntry* Window::entry_variable
```

GtkEntry to set the variable name.

Definition at line 176 of file interface.h.

3.8.2.30 entry_variables

```
GtkEntry* Window::entry_variables
```

Variables file GtkEntry.

Definition at line 98 of file interface.h.

3.8.2.31 experiment

```
Experiment* Window::experiment
```

Array of experiments data.

Definition at line 213 of file interface.h.

3.8.2.32 frame_algorithm

```
GtkFrame* Window::frame_algorithm
```

GtkFrame to set the algorithm.

Definition at line 112 of file interface.h.

3.8.2.33 frame_experiment

```
GtkFrame* Window::frame_experiment
```

Experiment GtkFrame.

Definition at line 198 of file interface.h.

3.8.2.34 frame_norm

```
GtkFrame* Window::frame_norm
```

GtkFrame to set the error norm.

Definition at line 99 of file interface.h.

3.8.2.35 frame_variable

```
GtkFrame* Window::frame_variable
```

Variable GtkFrame.

Definition at line 169 of file interface.h.

3.8.2.36 grid

```
GtkGrid* Window::grid
```

Main GtkGrid.

Definition at line 81 of file interface.h.

3.8.2.37 grid_algorithm

```
GtkGrid* Window::grid_algorithm
```

GtkGrid to set the algorithm.

Definition at line 113 of file interface.h.

3.8.2.38 grid_climbing

```
GtkGrid* Window::grid_climbing
```

GtkGrid to pack the hill climbing method widgets.

Definition at line 147 of file interface.h.

3.8.2.39 grid_experiment

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

Experiment GtkGrid.

Definition at line 199 of file interface.h.

3.8.2.40 grid_files

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

Files GtkGrid.

Definition at line 90 of file interface.h.

3.8.2.41 grid_norm

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

GtkGrid to set the error norm.

Definition at line 100 of file interface.h.

3.8.2.42 grid_variable

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

Variable GtkGrid.

Definition at line 170 of file interface.h.

3.8.2.43 id_experiment

```
gulong Window::id_experiment
```

Identifier of the combo_experiment signal.

Definition at line 216 of file interface.h.

3.8.2.44 id_experiment_name

```
gulong Window::id_experiment_name
```

Identifier of the button_experiment signal.

Definition at line 217 of file interface.h.

3.8.2.45 id_input

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

Array of identifiers of the button_template signal.

Definition at line 222 of file interface.h.

3.8.2.46 id_template

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

Array of identifiers of the check_template signal.

Definition at line 220 of file interface.h.

3.8.2.47 id_variable

```
gulong Window::id_variable
```

Identifier of the combo_variable signal.

Definition at line 218 of file interface.h.

3.8.2.48 id_variable_label

```
gulong Window::id_variable_label
```

Identifier of the entry_variable signal.

Definition at line 219 of file interface.h.

3.8.2.49 label_adaptation

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

GtkLabel to set the adaptation ratio.

Definition at line 142 of file interface.h.

3.8.2.50 label_bests

```
GtkLabel* Window::label_bests
```

GtkLabel to set the best number.

Definition at line 129 of file interface.h.

3.8.2.51 label_bits

```
GtkLabel* Window::label_bits
```

Bits number GtkLabel.

Definition at line 193 of file interface.h.

3.8.2.52 label_estimates

```
GtkLabel* Window::label_estimates
```

GtkLabel to set the estimates number.

Definition at line 158 of file interface.h.

3.8.2.53 label_experiment

```
GtkLabel* Window::label_experiment
```

Experiment GtkLabel.

Definition at line 203 of file interface.h.

3.8.2.54 label_generations

```
GtkLabel* Window::label_generations
```

GtkLabel to set the generations number.

Definition at line 134 of file interface.h.

3.8.2.55 label_iterations

```
GtkLabel* Window::label_iterations
```

GtkLabel to set the iterations number.

Definition at line 124 of file interface.h.

3.8.2.56 label_max

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

Maximum GtkLabel.

Definition at line 180 of file interface.h.

3.8.2.57 label_min

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

Minimum GtkLabel.

Definition at line 177 of file interface.h.

3.8.2.58 label_mutation

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

GtkLabel to set the mutation ratio.

Definition at line 137 of file interface.h.

3.8.2.59 label_p

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

GtkLabel to set the p parameter.

Definition at line 108 of file interface.h.

3.8.2.60 label_population

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

GtkLabel to set the population number.

Definition at line 131 of file interface.h.

3.8.2.61 label_precision

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

Precision GtkLabel.

Definition at line 189 of file interface.h.

3.8.2.62 label_relaxation

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

GtkLabel to set the relaxation parameter.

Definition at line 161 of file interface.h.

3.8.2.63 label_reproduction

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

GtkLabel to set the reproduction ratio.

Definition at line 139 of file interface.h.

3.8.2.64 label_result

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

Result file GtkLabel.

Definition at line 95 of file interface.h.

3.8.2.65 label_simulations

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

GtkLabel to set the simulations number.

Definition at line 121 of file interface.h.

3.8.2.66 label_simulator

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

Simulator program GtkLabel.

Definition at line 91 of file interface.h.

3.8.2.67 label_step

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

GtkLabel to set the step.

Definition at line 195 of file interface.h.

3.8.2.68 label_steps

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

GtkLabel to set the steps number.

Definition at line 156 of file interface.h.

3.8.2.69 label_sweeps

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

Sweeps number GtkLabel.

Definition at line 191 of file interface.h.

3.8.2.70 label_threshold

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

GtkLabel to set the threshold.

Definition at line 165 of file interface.h.

3.8.2.71 label_tolerance

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

GtkLabel to set the tolerance.

Definition at line 127 of file interface.h.

3.8.2.72 label_variable

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

Variable GtkLabel.

Definition at line 175 of file interface.h.

3.8.2.73 label_variables

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

Variables file GtkLabel.

Definition at line 97 of file interface.h.

3.8.2.74 label_weight

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

Weight GtkLabel.

Definition at line 206 of file interface.h.

3.8.2.75 logo

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

Logo GdkPixbuf.

Definition at line 212 of file interface.h.

3.8.2.76 nexperiments

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

Number of experiments.

Definition at line 224 of file interface.h.

3.8.2.77 nvariables

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

Number of variables.

Definition at line 225 of file interface.h.

3.8.2.78 scrolled_max

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

Maximum GtkScrolledWindow.

Definition at line 182 of file interface.h.

3.8.2.79 scrolled_maxabs

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

Absolute maximum GtkScrolledWindow.

Definition at line 188 of file interface.h.

3.8.2.80 scrolled_min

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

Minimum GtkScrolledWindow.

Definition at line 179 of file interface.h.

3.8.2.81 scrolled_minabs

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

Absolute minimum GtkScrolledWindow.

Definition at line 185 of file interface.h.

3.8.2.82 scrolled_p

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

GtkScrolledWindow to set the p parameter.

Definition at line 110 of file interface.h.

3.8.2.83 scrolled_step

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

step GtkScrolledWindow.

Definition at line 197 of file interface.h.

3.8.2.84 scrolled_threshold

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

GtkScrolledWindow to set the threshold.

Definition at line 167 of file interface.h.

3.8.2.85 spin_adaptation

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

GtkSpinButton to set the adaptation ratio.

Definition at line 143 of file interface.h.

3.8.2.86 spin_bests

```
GtkSpinButton* Window::spin_bests
```

GtkSpinButton to set the best number.

Definition at line 130 of file interface.h.

3.8.2.87 spin_bits

```
GtkSpinButton* Window::spin_bits
```

Bits number GtkSpinButton.

Definition at line 194 of file interface.h.

3.8.2.88 spin_estimates

```
GtkSpinButton* Window::spin_estimates
```

GtkSpinButton to set the estimates number.

Definition at line 159 of file interface.h.

3.8.2.89 spin_generations

```
GtkSpinButton* Window::spin_generations
```

GtkSpinButton to set the generations number.

Definition at line 135 of file interface.h.

3.8.2.90 spin_iterations

```
GtkSpinButton* Window::spin_iterations
```

GtkSpinButton to set the iterations number.

Definition at line 125 of file interface.h.

3.8.2.91 spin_max

```
GtkSpinButton* Window::spin_max
```

Maximum GtkSpinButton.

Definition at line 181 of file interface.h.

3.8.2.92 spin_maxabs

```
GtkSpinButton* Window::spin_maxabs
```

Absolute maximum GtkSpinButton.

Definition at line 187 of file interface.h.

3.8.2.93 spin_min

```
GtkSpinButton* Window::spin_min
```

Minimum GtkSpinButton.

Definition at line 178 of file interface.h.

3.8.2.94 spin_minabs

```
GtkSpinButton* Window::spin_minabs
```

Absolute minimum GtkSpinButton.

Definition at line 184 of file interface.h.

3.8.2.95 spin_mutation

```
GtkSpinButton* Window::spin_mutation
```

GtkSpinButton to set the mutation ratio.

Definition at line 138 of file interface.h.

3.8.2.96 spin_p

```
GtkSpinButton* Window::spin_p
```

GtkSpinButton to set the p parameter.

Definition at line 109 of file interface.h.

3.8.2.97 spin_population

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

GtkSpinButton to set the population number.

Definition at line 132 of file interface.h.

3.8.2.98 spin_precision

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

Precision digits GtkSpinButton.

Definition at line 190 of file interface.h.

3.8.2.99 spin_relaxation

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

GtkSpinButton to set the relaxation parameter.

Definition at line 163 of file interface.h.

3.8.2.100 spin_reproduction

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

GtkSpinButton to set the reproduction ratio.

Definition at line 140 of file interface.h.

3.8.2.101 spin_simulations

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

GtkSpinButton to set the simulations number.

Definition at line 122 of file interface.h.

3.8.2.102 spin_step

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

GtkSpinButton to set the step.

Definition at line 196 of file interface.h.

3.8.2.103 spin_steps

```
GtkSpinButton* Window::spin_steps
```

GtkSpinButton to set the steps number.

Definition at line 157 of file interface.h.

3.8.2.104 spin_sweeps

```
GtkSpinButton* Window::spin_sweeps
```

Sweeps number GtkSpinButton.

Definition at line 192 of file interface.h.

3.8.2.105 spin_threshold

```
GtkSpinButton* Window::spin_threshold
```

GtkSpinButton to set the threshold.

Definition at line 166 of file interface.h.

3.8.2.106 spin_tolerance

```
GtkSpinButton* Window::spin_tolerance
```

GtkSpinButton to set the tolerance.

Definition at line 128 of file interface.h.

3.8.2.107 spin_weight

```
GtkSpinButton* Window::spin_weight
```

Weight GtkSpinButton.

Definition at line 207 of file interface.h.

3.8.2.108 variable

```
Variable* Window::variable
```

Array of variables data.

Definition at line 214 of file interface.h.

3.8.2.109 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

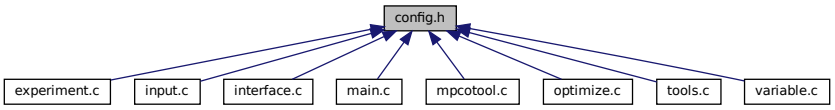
Chapter 4

File Documentation

4.1 config.h File Reference

Configuration header file.

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



Macros

- #define HAVE_MPI 1
- #define MAX_NINPUTS 8

  Maximum number of input files in the simulator program.
- #define NALGORITHMS 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_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_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_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 HAVE_MPI

```
#define HAVE_MPI 1
```

Definition at line 42 of file config.h.

4.1.2.5 LABEL_ABSOLUTE_MAXIMUM

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

absolute maximum label.

Definition at line 69 of file config.h.

4.1.2.6 LABEL_ABSOLUTE_MINIMUM

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

absolute minimum label.

Definition at line 67 of file config.h.

4.1.2.7 LABEL_ADAPTATION

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

adaption label.

Definition at line 70 of file config.h.

4.1.2.8 LABEL_ALGORITHM

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

algoritm label.

Definition at line 71 of file config.h.

4.1.2.9 LABEL_CLIMBING

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

climbing label.

Definition at line 72 of file config.h.

4.1.2.10 LABEL_COORDINATES

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

coordinates label.

Definition at line 73 of file config.h.

4.1.2.11 LABEL_EUCLIDIAN

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

euclidian label.

Definition at line 74 of file config.h.

4.1.2.12 LABEL_EVALUATOR

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

evaluator label.

Definition at line 75 of file config.h.

4.1.2.13 LABEL_EXPERIMENT

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

experiment label.

Definition at line 76 of file config.h.

4.1.2.14 LABEL_EXPERIMENTS

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

experiment label.

Definition at line 77 of file config.h.

4.1.2.15 LABEL_GENETIC

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

genetic label.

Definition at line 78 of file config.h.

4.1.2.16 LABEL_MAXIMUM

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

maximum label.

Definition at line 80 of file config.h.

4.1.2.17 LABEL_MINIMUM

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

minimum label.

Definition at line 79 of file config.h.

4.1.2.18 LABEL_MONTE_CARLO

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

Monte-Carlo label.

Definition at line 81 of file config.h.

4.1.2.19 LABEL_MUTATION

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

mutation label.

Definition at line 82 of file config.h.

4.1.2.20 LABEL_NAME

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

name label.

Definition at line 83 of file config.h.

4.1.2.21 LABEL_NBEST

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

nbest label.

Definition at line 84 of file config.h.

4.1.2.22 LABEL_NBITS

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

nbits label.

Definition at line 85 of file config.h.

4.1.2.23 LABEL_NESTIMATES

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

nestimates label.

Definition at line 86 of file config.h.

4.1.2.24 LABEL_NGENERATIONS

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

ngenerations label.

Definition at line 87 of file config.h.

4.1.2.25 LABEL_NITERATIONS

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

niterations label.

Definition at line 88 of file config.h.

4.1.2.26 LABEL_NORM

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

norm label.

Definition at line 89 of file config.h.

4.1.2.27 LABEL_NPOPULATION

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

npopulation label.

Definition at line 90 of file config.h.

4.1.2.28 LABEL_NSIMULATIONS

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

nsimulations label.

Definition at line 91 of file config.h.

4.1.2.29 LABEL_NSTEPS

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

nsteps label.

Definition at line 92 of file config.h.

4.1.2.30 LABEL_NSWEEPS

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

nsweeps label.

Definition at line 93 of file config.h.

4.1.2.31 LABEL_OPTIMIZE

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

optimize label.

Definition at line 94 of file config.h.

4.1.2.32 LABEL_ORTHOGONAL

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

orthogonal label.

Definition at line 95 of file config.h.

4.1.2.33 LABEL_P

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

p label.

Definition at line 96 of file config.h.

4.1.2.34 LABEL_PRECISION

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

precision label.

Definition at line 97 of file config.h.

4.1.2.35 LABEL_RANDOM

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

random label.

Definition at line 98 of file config.h.

4.1.2.36 LABEL_RELAXATION

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

relaxation label.

Definition at line 99 of file config.h.

4.1.2.37 LABEL_REPRODUCTION

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

reproduction label.

Definition at line 100 of file config.h.

4.1.2.38 LABEL_RESULT_FILE

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

result_file label.

Definition at line 101 of file config.h.

4.1.2.39 LABEL_SEED

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

seed label.

Definition at line 103 of file config.h.

4.1.2.40 LABEL_SIMULATOR

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

simulator label.

Definition at line 102 of file config.h.

4.1.2.41 LABEL_STEP

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

step label.

Definition at line 104 of file config.h.

4.1.2.42 LABEL_SWEEP

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

sweep label.

Definition at line 105 of file config.h.

4.1.2.43 LABEL_TAXICAB

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

taxicab label.

Definition at line 106 of file config.h.

4.1.2.44 LABEL_TEMPLATE1

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

template1 label.

Definition at line 107 of file config.h.

4.1.2.45 LABEL_TEMPLATE2

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

template2 label.

Definition at line 108 of file config.h.

4.1.2.46 LABEL_TEMPLATE3

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

template3 label.

Definition at line 109 of file config.h.

4.1.2.47 LABEL_TEMPLATE4

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

template4 label.

Definition at line 110 of file config.h.

4.1.2.48 LABEL_TEMPLATE5

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

template5 label.

Definition at line 111 of file config.h.

4.1.2.49 LABEL_TEMPLATE6

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

template6 label.

Definition at line 112 of file config.h.

4.1.2.50 LABEL_TEMPLATE7

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

template7 label.

Definition at line 113 of file config.h.

4.1.2.51 LABEL_TEMPLATE8

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

template8 label.

Definition at line 114 of file config.h.

4.1.2.52 LABEL_THRESHOLD

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

threshold label.

Definition at line 115 of file config.h.

4.1.2.53 LABEL_TOLERANCE

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

tolerance label.

Definition at line 116 of file config.h.

4.1.2.54 LABEL_VARIABLE

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

variable label.

Definition at line 117 of file config.h.

4.1.2.55 LABEL_VARIABLES

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

variables label.

Definition at line 118 of file config.h.

4.1.2.56 LABEL_VARIABLES_FILE

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

variables label.

Definition at line 119 of file config.h.

4.1.2.57 LABEL_WEIGHT

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

weight label.

Definition at line 120 of file config.h.

4.1.2.58 LOCALE_DIR

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

Locales directory.

Definition at line 61 of file config.h.

4.1.2.59 MAX_NINPUTS

```
#define MAX_NINPUTS 8
```

Maximum number of input files in the simulator program.

Definition at line 47 of file config.h.

4.1.2.60 NALGORITHMS

```
#define NALGORITHMS 4
```

Number of stochastic algorithms.

Definition at line 48 of file config.h.

4.1.2.61 NCLIMBINGS

```
#define NCLIMBINGS 2
```

Number of hill climbing estimate methods.

Definition at line 49 of file config.h.

4.1.2.62 NNORMS

```
#define NNORMS 4
```

Number of error norms.

Definition at line 50 of file config.h.

4.1.2.63 NPRECISIONS

```
#define NPRECISIONS 15
```

Number of precisions.

Definition at line 51 of file config.h.

4.1.2.64 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 125 of file config.h.

```
00126 {
00127   INPUT_TYPE_XML = 0,
00128   INPUT_TYPE_JSON = 1
00129 };
```

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
00039 #ifndef CONFIG__H
00040 #define CONFIG__H 1
00041
00042 #define HAVE_MPI 1
00043
00044 // Array sizes
00045
00046 #define MAX_NINPUTS 8
00048 #define NALGORITHMS 4
00049 #define NCLIMBINGS 2
00050 #define NNORMS 4
00051 #define NPRECISIONS 15
00052
00053 // Default choices
00054
00055 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00056 #define DEFAULT_RANDOM_SEED 7007
00057 #define DEFAULT_RELAXATION 1.
00058
00059 // Interface labels
00060
00061 #define LOCALE_DIR "locales"
00062 #define PROGRAM_INTERFACE "mpcotool"
00063
00064 // Labels
00065
00066 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00068 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00070 #define LABEL_ADAPTATION "adaptation"
00071 #define LABEL_ALGORITHM "algorithm"
00072 #define LABEL_CLIMBING "climbing"
00073 #define LABEL_COORDINATES "coordinates"
00074 #define LABEL_EUCLIDIAN "euclidian"
00075 #define LABEL_EVALUATOR "evaluator"
00076 #define LABEL_EXPERIMENT "experiment"
00077 #define LABEL_EXPERIMENTS "experiments"
00078 #define LABEL_GENETIC "genetic"
00079 #define LABEL_MINIMUM "minimum"
00080 #define LABEL_MAXIMUM "maximum"
00081 #define LABEL_MONTE_CARLO "Monte-Carlo"
00082 #define LABEL_MUTATION "mutation"
00083 #define LABEL_NAME "name"
00084 #define LABEL_NBEST "nbest"
00085 #define LABEL_NBITS "nbits"
00086 #define LABEL_NESTIMATES "nestimates"
00087 #define LABEL_NGENERATIONS "ngenerations"
00088 #define LABEL_NITERATIONS "niterations"
00089 #define LABEL_NORM "norm"
00090 #define LABEL_NPOPULATION "npopulation"
00091 #define LABEL_NSIMULATIONS "nsimulations"
00092 #define LABEL_NSTEPS "nsteps"
00093 #define LABEL_NSWEEPS "nsweeps"
00094 #define LABEL_OPTIMIZE "optimize"
00095 #define LABEL_ORTHOGONAL "orthogonal"
00096 #define LABEL_P "p"
00097 #define LABEL_PRECISION "precision"
00098 #define LABEL_RANDOM "random"
00099 #define LABEL_RELAXATION "relaxation"
00100 #define LABEL_REPRODUCTION "reproduction"
00101 #define LABEL_RESULT_FILE "result_file"
00102 #define LABEL_SIMULATOR "simulator"
00103 #define LABEL_SEED "seed"
```

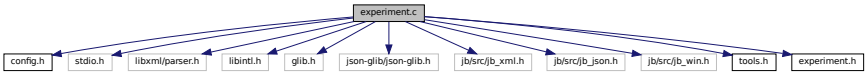
```
00104 #define LABEL_STEP "step"
00105 #define LABEL_SWEEP "sweep"
00106 #define LABEL_TAXICAB "taxicab"
00107 #define LABEL_TEMPLATE1 "template1"
00108 #define LABEL_TEMPLATE2 "template2"
00109 #define LABEL_TEMPLATE3 "template3"
00110 #define LABEL_TEMPLATE4 "template4"
00111 #define LABEL_TEMPLATE5 "template5"
00112 #define LABEL_TEMPLATE6 "template6"
00113 #define LABEL_TEMPLATE7 "template7"
00114 #define LABEL_TEMPLATE8 "template8"
00115 #define LABEL_THRESHOLD "threshold"
00116 #define LABEL_TOLERANCE "tolerance"
00117 #define LABEL_VARIABLE "variable"
00118 #define LABEL_VARIABLES "variables"
00119 #define LABEL_VARIABLES_FILE "variables_file"
00120 #define LABEL_WEIGHT "weight"
00121
00122 // Enumerations
00123
00125 enum INPUT_TYPE
00126 {
00127   INPUT_TYPE_XML = 0,
00128   INPUT_TYPE_JSON = 1
00129 };
00130
00131 #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 xmlChar strings with stencil labels.

4.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

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

experiment	Experiment struct.
message	Error message.

Definition at line 112 of file experiment.c.

```
00114 {
00115   char buffer[64];
00116   if (!experiment->name)
00117     snprintf (buffer, 64, "%s:  %s", _("Experiment"), message);
00118   else
00119     snprintf (buffer, 64, "%s %s:  %s", _("Experiment"),
00120               experiment->name, message);
00121   error_message = g_strdup (buffer);
00122 }
```

4.3.3.2 experiment_free()

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

Function to free the memory of an Experiment struct.

Parameters

experiment	Experiment struct.
type	Type of input file.

Definition at line 83 of file experiment.c.

```
00085 {
00086   unsigned int i;
00087 #if DEBUG_EXPERIMENT
00088   fprintf (stderr, "experiment_free:  start\n");
00089 #endif
00090   if (type == INPUT_TYPE_XML)
00091     {
00092       for (i = 0; i < experiment->ninputs; ++i)
00093         xmlFree (experiment->stencil[i]);
00094       xmlFree (experiment->name);
00095     }
00096   else
00097     {
00098       for (i = 0; i < experiment->ninputs; ++i)
00099         g_free (experiment->stencil[i]);
00100       g_free (experiment->name);
00101     }
00102   experiment->ninputs = 0;
00103 #if DEBUG_EXPERIMENT
00104   fprintf (stderr, "experiment_free:  end\n");
00105 #endif
00106 }
```

4.3.3.3 experiment_new()

```
static void experiment_new (
             Experiment * experiment )  [static]
```

Function to create a new Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 64 of file experiment.c.

```
00065 {
00066   unsigned int i;
00067 #if DEBUG_EXPERIMENT
00068   fprintf (stderr, "experiment_new:  start\n");
00069 #endif
00070   experiment->name = NULL;
00071   experiment->ninputs = 0;
00072   for (i = 0; i < MAX_NINPUTS; ++i)
00073     experiment->stencil[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075   fprintf (stderr, "input_new:  end\n");
00076 #endif
00077 }
```

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

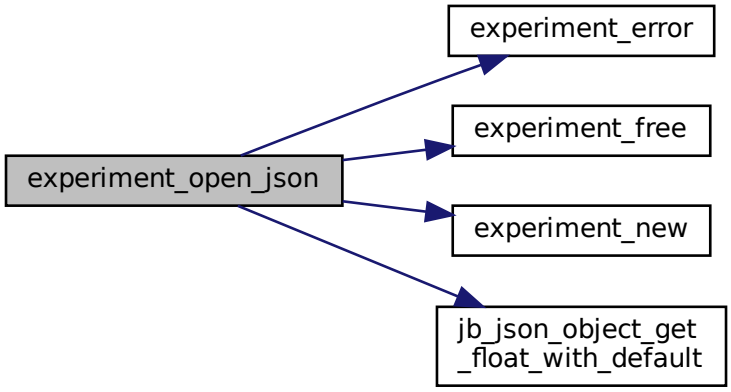
experiment	Experiment struct.
node	JSON node.
ninputs	Number of the simulator input files.

Definition at line 233 of file experiment.c.

```
00237 {
00238   char buffer[64];
00239   JsonObject *object;
00240   const char *name;
00241   int error_code;
00242   unsigned int i;
00243
00244 #if DEBUG_EXPERIMENT
00245   fprintf (stderr, "experiment_open_json:  start\n");
00246 #endif
00247
00248   // Resetting experiment data
00249   experiment_new (experiment);
00250
00251   // Getting JSON object
00252   object = json_node_get_object (node);
00253
00254   // Reading the experimental data
00255   name = json_object_get_string_member (object, LABEL_NAME);
00256   if (!name)
00257     {
00258       experiment_error (experiment, _("no data file name"));
00259       goto exit_on_error;
00260     }
00261   experiment->name = g_strdup (name);
00262 #if DEBUG_EXPERIMENT
00263   fprintf (stderr, "experiment_open_json:  name=%s\n", experiment->name);
00264 #endif
00265   experiment->weight
00266     = jb_json_object_get_float_with_default (object, LABEL_WEIGHT, &error_code,
00267                                              1.);
00268   if (!error_code)
```

```
00269     {
00270       experiment_error (experiment, _("bad weight"));
00271       goto exit_on_error;
00272     }
00273 #if DEBUG_EXPERIMENT
00274   fprintf (stderr, "experiment_open_json:  weight=%lg\n", experiment->weight);
00275 #endif
00276   name = json_object_get_string_member (object, stencil[0]);
00277   if (name)
00278     {
00279 #if DEBUG_EXPERIMENT
00280       fprintf (stderr, "experiment_open_json:  experiment=%s template1=%s\n",
00281                name, stencil[0]);
00282 #endif
00283       ++experiment->ninputs;
00284     }
00285   else
00286     {
00287       experiment_error (experiment, _("no template"));
00288       goto exit_on_error;
00289     }
00290   experiment->stencil[0] = g_strdup (name);
00291   for (i = 1; i < MAX_NINPUTS; ++i)
00292     {
00293 #if DEBUG_EXPERIMENT
00294       fprintf (stderr, "experiment_open_json:  stencil%u\n", i + 1);
00295 #endif
00296       if (json_object_get_member (object, stencil[i]))
00297         {
00298           if (ninputs && ninputs <= i)
00299             {
00300               experiment_error (experiment, _("bad templates number"));
00301               goto exit_on_error;
00302             }
00303           name = json_object_get_string_member (object, stencil[i]);
00304 #if DEBUG_EXPERIMENT
00305           fprintf (stderr,
00306                    "experiment_open_json:  experiment=%s stencil%u=%s\n",
00307                    experiment->nexperiments, name, stencil[i]);
00308 #endif
00309           experiment->stencil[i] = g_strdup (name);
00310           ++experiment->ninputs;
00311         }
00312       else if (ninputs && ninputs > i)
00313         {
00314           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00315           experiment_error (experiment, buffer);
00316           goto exit_on_error;
00317         }
00318       else
00319         break;
00320     }
00321
00322 #if DEBUG_EXPERIMENT
00323   fprintf (stderr, "experiment_open_json:  end\n");
00324 #endif
00325   return 1;
00326
00327 exit_on_error:
00328   experiment_free (experiment, INPUT_TYPE_JSON);
00329 #if DEBUG_EXPERIMENT
00330   fprintf (stderr, "experiment_open_json:  end\n");
00331 #endif
00332   return 0;
00333 }
```

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

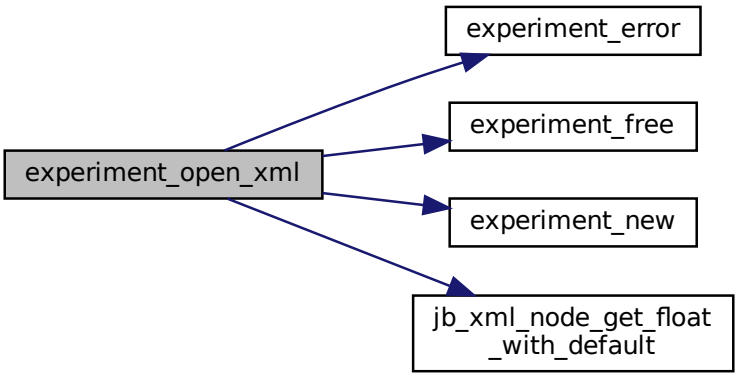
experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Definition at line 130 of file experiment.c.

```
00134 {
00135   char buffer[64];
00136   int error_code;
00137   unsigned int i;
00138
00139 #if DEBUG_EXPERIMENT
00140   fprintf (stderr, "experiment_open_xml:  start\n");
00141 #endif
00142
00143   // Resetting experiment data
00144   experiment_new (experiment);
00145
```

```
00146   // Reading the experimental data
00147   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00148   if (!experiment->name)
00149     {
00150       experiment_error (experiment, _("no data file name"));
00151       goto exit_on_error;
00152     }
00153 #if DEBUG_EXPERIMENT
00154   fprintf (stderr, "experiment_open_xml:  name=%s\n", experiment->name);
00155 #endif
00156   experiment->weight
00157     = jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_WEIGHT,
00158                                           &error_code, 1.);
00159   if (!error_code)
00160     {
00161       experiment_error (experiment, _("bad weight"));
00162       goto exit_on_error;
00163     }
00164 #if DEBUG_EXPERIMENT
00165   fprintf (stderr, "experiment_open_xml:  weight=%lg\n", experiment->weight);
00166 #endif
00167   experiment->stencil[0]
00168     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00169   if (experiment->stencil[0])
00170     {
00171 #if DEBUG_EXPERIMENT
00172       fprintf (stderr, "experiment_open_xml:  experiment=%s stencil1=%s\n",
00173                experiment->name, stencil[0]);
00174 #endif
00175       ++experiment->ninputs;
00176     }
00177   else
00178     {
00179       experiment_error (experiment, _("no template"));
00180       goto exit_on_error;
00181     }
00182   for (i = 1; i < MAX_NINPUTS; ++i)
00183     {
00184 #if DEBUG_EXPERIMENT
00185       fprintf (stderr, "experiment_open_xml:  stencil%u\n", i + 1);
00186 #endif
00187       if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00188         {
00189           if (ninputs && ninputs <= i)
00190             {
00191               experiment_error (experiment, _("bad templates number"));
00192               goto exit_on_error;
00193             }
00194           experiment->stencil[i]
00195             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00196 #if DEBUG_EXPERIMENT
00197           fprintf (stderr,
00198                    "experiment_open_xml:  experiment=%s stencil%u=%s\n",
00199                    experiment->nexperiments, experiment->name,
00200                    experiment->stencil[i]);
00201 #endif
00202           ++experiment->ninputs;
00203         }
00204       else if (ninputs && ninputs > i)
00205         {
00206           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00207           experiment_error (experiment, buffer);
00208           goto exit_on_error;
00209         }
00210       else
00211         break;
00212     }
00213
00214 #if DEBUG_EXPERIMENT
00215   fprintf (stderr, "experiment_open_xml:  end\n");
00216 #endif
00217   return 1;
00218
00219 exit_on_error:
00220   experiment_free (experiment, INPUT_TYPE_XML);
00221 #if DEBUG_EXPERIMENT
00222   fprintf (stderr, "experiment_open_xml:  end\n");
00223 #endif
00224   return 0;
00225 }
```

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 xmlChar strings with stencil labels.

Definition at line 53 of file experiment.c.

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
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "jb/src/jb_xml.h"
00046 #include "jb/src/jb_json.h"
00047 #include "jb/src/jb_win.h"
00048 #include "tools.h"
00049 #include "experiment.h"
00050
00051 #define DEBUG_EXPERIMENT 0
00052
00053 const char *stencil[MAX_NINPUTS] = {
00054   LABEL_TEMPLATE1, LABEL_TEMPLATE2, LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00055   LABEL_TEMPLATE5, LABEL_TEMPLATE6, LABEL_TEMPLATE7, LABEL_TEMPLATE8
00056 };
00057
00059
00063 static void
00064 experiment_new (Experiment * experiment)
00065 {
00066   unsigned int i;
00067 #if DEBUG_EXPERIMENT
00068   fprintf (stderr, "experiment_new:  start\n");
00069 #endif
00070   experiment->name = NULL;
00071   experiment->ninputs = 0;
00072   for (i = 0; i < MAX_NINPUTS; ++i)
00073     experiment->stencil[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075   fprintf (stderr, "input_new:  end\n");
00076 #endif
00077 }
00078
00082 void
00083 experiment_free (Experiment * experiment,
00084                  unsigned int type)
00085 {
00086   unsigned int i;
00087 #if DEBUG_EXPERIMENT
00088   fprintf (stderr, "experiment_free:  start\n");
00089 #endif
00090   if (type == INPUT_TYPE_XML)
00091     {
00092       for (i = 0; i < experiment->ninputs; ++i)
00093         xmlFree (experiment->stencil[i]);
00094       xmlFree (experiment->name);
00095     }
00096   else
00097     {
00098       for (i = 0; i < experiment->ninputs; ++i)
00099         g_free (experiment->stencil[i]);
00100       g_free (experiment->name);
00101     }
00102   experiment->ninputs = 0;
00103 #if DEBUG_EXPERIMENT
00104   fprintf (stderr, "experiment_free:  end\n");
00105 #endif
00106 }
00107
00111 void
00112 experiment_error (Experiment * experiment,
00113                   char *message)
00114 {
00115   char buffer[64];
00116   if (!experiment->name)
00117     snprintf (buffer, 64, "%s:  %s", _("Experiment"), message);
00118   else
00119     snprintf (buffer, 64, "%s %s:  %s", _("Experiment"),
```

```
00120               experiment->name, message);
00121   error_message = g_strdup (buffer);
00122 }
00123
00129 int
00130 experiment_open_xml (Experiment * experiment,
00131                      xmlNode * node,
00132                      unsigned int ninputs)
00134 {
00135   char buffer[64];
00136   int error_code;
00137   unsigned int i;
00138
00139 #if DEBUG_EXPERIMENT
00140   fprintf (stderr, "experiment_open_xml:  start\n");
00141 #endif
00142
00143   // Resetting experiment data
00144   experiment_new (experiment);
00145
00146   // Reading the experimental data
00147   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00148   if (!experiment->name)
00149     {
00150       experiment_error (experiment, _("no data file name"));
00151       goto exit_on_error;
00152     }
00153 #if DEBUG_EXPERIMENT
00154   fprintf (stderr, "experiment_open_xml:  name=%s\n", experiment->name);
00155 #endif
00156   experiment->weight
00157     = jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_WEIGHT,
00158                                           &error_code, 1.);
00159   if (!error_code)
00160     {
00161       experiment_error (experiment, _("bad weight"));
00162       goto exit_on_error;
00163     }
00164 #if DEBUG_EXPERIMENT
00165   fprintf (stderr, "experiment_open_xml:  weight=%lg\n", experiment->weight);
00166 #endif
00167   experiment->stencil[0]
00168     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00169   if (experiment->stencil[0])
00170     {
00171 #if DEBUG_EXPERIMENT
00172       fprintf (stderr, "experiment_open_xml:  experiment=%s stencil1=%s\n",
00173                experiment->name, stencil[0]);
00174 #endif
00175       ++experiment->ninputs;
00176     }
00177   else
00178     {
00179       experiment_error (experiment, _("no template"));
00180       goto exit_on_error;
00181     }
00182   for (i = 1; i < MAX_NINPUTS; ++i)
00183     {
00184 #if DEBUG_EXPERIMENT
00185       fprintf (stderr, "experiment_open_xml:  stencil%u\n", i + 1);
00186 #endif
00187       if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00188         {
00189           if (ninputs && ninputs <= i)
00190             {
00191               experiment_error (experiment, _("bad templates number"));
00192               goto exit_on_error;
00193             }
00194           experiment->stencil[i]
00195             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00196 #if DEBUG_EXPERIMENT
00197           fprintf (stderr,
00198                    "experiment_open_xml:  experiment=%s stencil%u=%s\n",
00199                    experiment->nexperiments, experiment->name,
00200                    experiment->stencil[i]);
00201 #endif
00202           ++experiment->ninputs;
00203         }
00204       else if (ninputs && ninputs > i)
00205         {
00206           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00207           experiment_error (experiment, buffer);
00208           goto exit_on_error;
00209         }
00210       else
00211         break;
00212     }
```

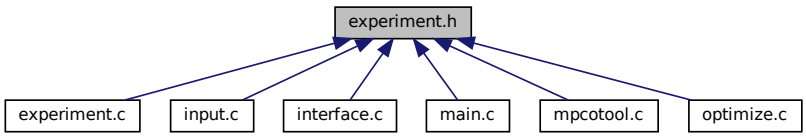
```
00213
00214 #if DEBUG_EXPERIMENT
00215   fprintf (stderr, "experiment_open_xml:  end\n");
00216 #endif
00217   return 1;
00218
00219 exit_on_error:
00220   experiment_free (experiment, INPUT_TYPE_XML);
00221 #if DEBUG_EXPERIMENT
00222   fprintf (stderr, "experiment_open_xml:  end\n");
00223 #endif
00224   return 0;
00225 }
00226
00232 int
00233 experiment_open_json (Experiment * experiment,
00234                       JsonNode * node,
00235                       unsigned int ninputs)
00237 {
00238   char buffer[64];
00239   JsonObject *object;
00240   const char *name;
00241   int error_code;
00242   unsigned int i;
00243
00244 #if DEBUG_EXPERIMENT
00245   fprintf (stderr, "experiment_open_json:  start\n");
00246 #endif
00247
00248   // Resetting experiment data
00249   experiment_new (experiment);
00250
00251   // Getting JSON object
00252   object = json_node_get_object (node);
00253
00254   // Reading the experimental data
00255   name = json_object_get_string_member (object, LABEL_NAME);
00256   if (!name)
00257     {
00258       experiment_error (experiment, _("no data file name"));
00259       goto exit_on_error;
00260     }
00261   experiment->name = g_strdup (name);
00262 #if DEBUG_EXPERIMENT
00263   fprintf (stderr, "experiment_open_json:  name=%s\n", experiment->name);
00264 #endif
00265   experiment->weight
00266     = jb_json_object_get_float_with_default (object, LABEL_WEIGHT, &error_code,
00267                                              1.);
00268   if (!error_code)
00269     {
00270       experiment_error (experiment, _("bad weight"));
00271       goto exit_on_error;
00272     }
00273 #if DEBUG_EXPERIMENT
00274   fprintf (stderr, "experiment_open_json:  weight=%lg\n", experiment->weight);
00275 #endif
00276   name = json_object_get_string_member (object, stencil[0]);
00277   if (name)
00278     {
00279 #if DEBUG_EXPERIMENT
00280       fprintf (stderr, "experiment_open_json:  experiment=%s template1=%s\n",
00281                name, stencil[0]);
00282 #endif
00283       ++experiment->ninputs;
00284     }
00285   else
00286     {
00287       experiment_error (experiment, _("no template"));
00288       goto exit_on_error;
00289     }
00290   experiment->stencil[0] = g_strdup (name);
00291   for (i = 1; i < MAX_NINPUTS; ++i)
00292     {
00293 #if DEBUG_EXPERIMENT
00294       fprintf (stderr, "experiment_open_json:  stencil%u\n", i + 1);
00295 #endif
00296       if (json_object_get_member (object, stencil[i]))
00297         {
00298           if (ninputs && ninputs <= i)
00299             {
00300               experiment_error (experiment, _("bad templates number"));
00301               goto exit_on_error;
00302             }
00303           name = json_object_get_string_member (object, stencil[i]);
00304 #if DEBUG_EXPERIMENT
00305           fprintf (stderr,
```

```
00306                   "experiment_open_json:  experiment=%s stencil%u=%s\n",
00307                   experiment->nexperiments, name, stencil[i]);
00308 #endif
00309           experiment->stencil[i] = g_strdup (name);
00310           ++experiment->ninputs;
00311         }
00312       else if (ninputs && ninputs > i)
00313         {
00314           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00315           experiment_error (experiment, buffer);
00316           goto exit_on_error;
00317         }
00318       else
00319         break;
00320     }
00321
00322 #if DEBUG_EXPERIMENT
00323   fprintf (stderr, "experiment_open_json:  end\n");
00324 #endif
00325   return 1;
00326
00327 exit_on_error:
00328   experiment_free (experiment, INPUT_TYPE_JSON);
00329 #if DEBUG_EXPERIMENT
00330   fprintf (stderr, "experiment_open_json:  end\n");
00331 #endif
00332   return 0;
00333 }
```

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 xmlChar strings with stencil labels.

4.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

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

experiment	Experiment struct.
message	Error message.

Definition at line 112 of file experiment.c.

```
00114 {
00115   char buffer[64];
00116   if (!experiment->name)
00117     snprintf (buffer, 64, "%s:  %s", _("Experiment"), message);
00118   else
00119     snprintf (buffer, 64, "%s %s:  %s", _("Experiment"),
00120               experiment->name, message);
00121   error_message = g_strdup (buffer);
00122 }
```

4.5.2.2 experiment_free()

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

Function to free the memory of an Experiment struct.

Parameters

experiment	Experiment struct.
type	Type of input file.

Definition at line 83 of file experiment.c.

```
00085 {
00086   unsigned int i;
00087 #if DEBUG_EXPERIMENT
00088   fprintf (stderr, "experiment_free:  start\n");
00089 #endif
00090   if (type == INPUT_TYPE_XML)
00091     {
00092       for (i = 0; i < experiment->ninputs; ++i)
00093         xmlFree (experiment->stencil[i]);
00094       xmlFree (experiment->name);
00095     }
00096   else
00097     {
00098       for (i = 0; i < experiment->ninputs; ++i)
00099         g_free (experiment->stencil[i]);
00100       g_free (experiment->name);
00101     }
00102   experiment->ninputs = 0;
00103 #if DEBUG_EXPERIMENT
00104   fprintf (stderr, "experiment_free:  end\n");
00105 #endif
00106 }
```

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

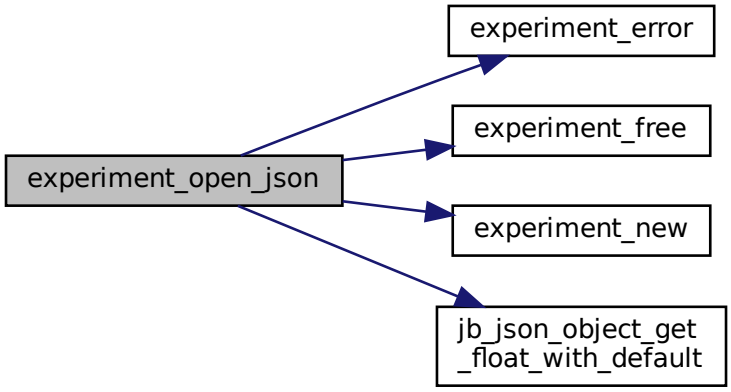
experiment	Experiment struct.
node	JSON node.
ninputs	Number of the simulator input files.

Definition at line 233 of file experiment.c.

```
00237 {
00238   char buffer[64];
00239   JsonObject *object;
00240   const char *name;
00241   int error_code;
00242   unsigned int i;
00243
00244 #if DEBUG_EXPERIMENT
00245   fprintf (stderr, "experiment_open_json:  start\n");
00246 #endif
00247
00248   // Resetting experiment data
00249   experiment_new (experiment);
```

```
00250
00251   // Getting JSON object
00252   object = json_node_get_object (node);
00253
00254   // Reading the experimental data
00255   name = json_object_get_string_member (object, LABEL_NAME);
00256   if (!name)
00257     {
00258       experiment_error (experiment, _("no data file name"));
00259       goto exit_on_error;
00260     }
00261   experiment->name = g_strdup (name);
00262 #if DEBUG_EXPERIMENT
00263   fprintf (stderr, "experiment_open_json:  name=%s\n", experiment->name);
00264 #endif
00265   experiment->weight
00266     = jb_json_object_get_float_with_default (object, LABEL_WEIGHT, &error_code,
00267                                              1.);
00268   if (!error_code)
00269     {
00270       experiment_error (experiment, _("bad weight"));
00271       goto exit_on_error;
00272     }
00273 #if DEBUG_EXPERIMENT
00274   fprintf (stderr, "experiment_open_json:  weight=%lg\n", experiment->weight);
00275 #endif
00276   name = json_object_get_string_member (object, stencil[0]);
00277   if (name)
00278     {
00279 #if DEBUG_EXPERIMENT
00280       fprintf (stderr, "experiment_open_json:  experiment=%s template1=%s\n",
00281                name, stencil[0]);
00282 #endif
00283       ++experiment->ninputs;
00284     }
00285   else
00286     {
00287       experiment_error (experiment, _("no template"));
00288       goto exit_on_error;
00289     }
00290   experiment->stencil[0] = g_strdup (name);
00291   for (i = 1; i < MAX_NINPUTS; ++i)
00292     {
00293 #if DEBUG_EXPERIMENT
00294       fprintf (stderr, "experiment_open_json:  stencil%u\n", i + 1);
00295 #endif
00296       if (json_object_get_member (object, stencil[i]))
00297         {
00298           if (ninputs && ninputs <= i)
00299             {
00300               experiment_error (experiment, _("bad templates number"));
00301               goto exit_on_error;
00302             }
00303           name = json_object_get_string_member (object, stencil[i]);
00304 #if DEBUG_EXPERIMENT
00305           fprintf (stderr,
00306                    "experiment_open_json:  experiment=%s stencil%u=%s\n",
00307                    experiment->nexperiments, name, stencil[i]);
00308 #endif
00309           experiment->stencil[i] = g_strdup (name);
00310           ++experiment->ninputs;
00311         }
00312       else if (ninputs && ninputs > i)
00313         {
00314           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00315           experiment_error (experiment, buffer);
00316           goto exit_on_error;
00317         }
00318       else
00319         break;
00320     }
00321
00322 #if DEBUG_EXPERIMENT
00323   fprintf (stderr, "experiment_open_json:  end\n");
00324 #endif
00325   return 1;
00326
00327 exit_on_error:
00328   experiment_free (experiment, INPUT_TYPE_JSON);
00329 #if DEBUG_EXPERIMENT
00330   fprintf (stderr, "experiment_open_json:  end\n");
00331 #endif
00332   return 0;
00333 }
```

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

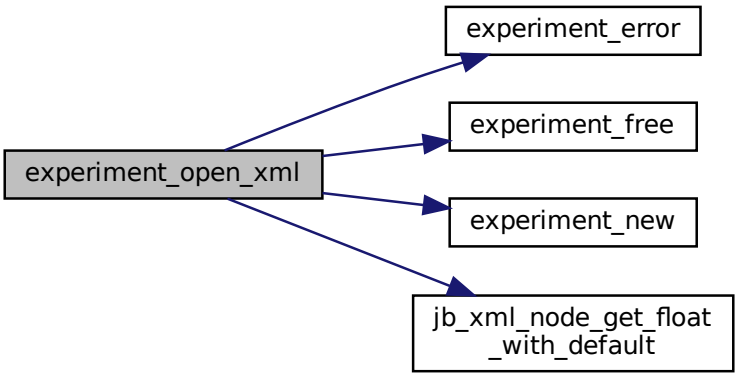
experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Definition at line 130 of file experiment.c.

```
00134 {
00135   char buffer[64];
00136   int error_code;
00137   unsigned int i;
00138
00139 #if DEBUG_EXPERIMENT
00140   fprintf (stderr, "experiment_open_xml:  start\n");
00141 #endif
00142
00143   // Resetting experiment data
00144   experiment_new (experiment);
00145
```

```
00146   // Reading the experimental data
00147   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00148   if (!experiment->name)
00149     {
00150       experiment_error (experiment, _("no data file name"));
00151       goto exit_on_error;
00152     }
00153 #if DEBUG_EXPERIMENT
00154   fprintf (stderr, "experiment_open_xml:  name=%s\n", experiment->name);
00155 #endif
00156   experiment->weight
00157     = jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_WEIGHT,
00158                                           &error_code, 1.);
00159   if (!error_code)
00160     {
00161       experiment_error (experiment, _("bad weight"));
00162       goto exit_on_error;
00163     }
00164 #if DEBUG_EXPERIMENT
00165   fprintf (stderr, "experiment_open_xml:  weight=%lg\n", experiment->weight);
00166 #endif
00167   experiment->stencil[0]
00168     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00169   if (experiment->stencil[0])
00170     {
00171 #if DEBUG_EXPERIMENT
00172       fprintf (stderr, "experiment_open_xml:  experiment=%s stencil1=%s\n",
00173                experiment->name, stencil[0]);
00174 #endif
00175       ++experiment->ninputs;
00176     }
00177   else
00178     {
00179       experiment_error (experiment, _("no template"));
00180       goto exit_on_error;
00181     }
00182   for (i = 1; i < MAX_NINPUTS; ++i)
00183     {
00184 #if DEBUG_EXPERIMENT
00185       fprintf (stderr, "experiment_open_xml:  stencil%u\n", i + 1);
00186 #endif
00187       if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00188         {
00189           if (ninputs && ninputs <= i)
00190             {
00191               experiment_error (experiment, _("bad templates number"));
00192               goto exit_on_error;
00193             }
00194           experiment->stencil[i]
00195             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00196 #if DEBUG_EXPERIMENT
00197           fprintf (stderr,
00198                    "experiment_open_xml:  experiment=%s stencil%u=%s\n",
00199                    experiment->nexperiments, experiment->name,
00200                    experiment->stencil[i]);
00201 #endif
00202           ++experiment->ninputs;
00203         }
00204       else if (ninputs && ninputs > i)
00205         {
00206           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00207           experiment_error (experiment, buffer);
00208           goto exit_on_error;
00209         }
00210       else
00211         break;
00212     }
00213
00214 #if DEBUG_EXPERIMENT
00215   fprintf (stderr, "experiment_open_xml:  end\n");
00216 #endif
00217   return 1;
00218
00219 exit_on_error:
00220   experiment_free (experiment, INPUT_TYPE_XML);
00221 #if DEBUG_EXPERIMENT
00222   fprintf (stderr, "experiment_open_xml:  end\n");
00223 #endif
00224   return 0;
00225 }
```

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 xmlChar strings with stencil labels.

Definition at line 53 of file experiment.c.

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,
```

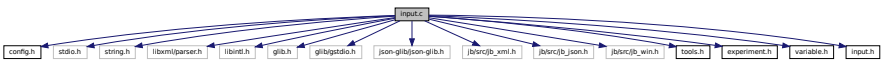
```
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (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
00038 #ifndef EXPERIMENT__H
00039 #define EXPERIMENT__H 1
00040
00045 typedef struct
00046 {
00047   char *name;
00048   char *stencil[MAX_NINPUTS];
00049   double weight;
00050   unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const char *stencil[MAX_NINPUTS];
00054
00055 // Public functions
00056 void experiment_free (Experiment * experiment, unsigned int type);
00057 void experiment_error (Experiment * experiment, char *message);
00058 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00059                          unsigned int ninputs);
00060 int experiment_open_json (Experiment * experiment, JsonNode * node,
00061                           unsigned int ninputs);
00062
00063 #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

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

message	Error message.

Definition at line 122 of file input.c.

```
00123 {
00124   char buffer[64];
00125   snprintf (buffer, 64, "%s:  %s\n", _("Input"), message);
00126   error_message = g_strdup (buffer);
00127 }
```

4.7.3.2 input_free()

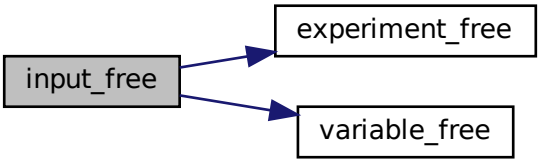
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 84 of file input.c.

```
00085 {
00086   unsigned int i;
00087 #if DEBUG_INPUT
00088   fprintf (stderr, "input_free:  start\n");
00089 #endif
00090   g_free (input->name);
00091   g_free (input->directory);
00092   for (i = 0; i < input->nexperiments; ++i)
00093     experiment_free (input->experiment + i, input->type);
00094   for (i = 0; i < input->nvariables; ++i)
00095     variable_free (input->variable + i, input->type);
00096   g_free (input->experiment);
00097   g_free (input->variable);
00098   if (input->type == INPUT_TYPE_XML)
00099     {
00100       xmlFree (input->evaluator);
00101       xmlFree (input->simulator);
00102       xmlFree (input->result);
00103       xmlFree (input->variables);
00104     }
00105   else
00106     {
00107       g_free (input->evaluator);
00108       g_free (input->simulator);
00109       g_free (input->result);
00110       g_free (input->variables);
00111     }
00112   input->nexperiments = input->nvariables = input->nsteps = 0;
00113 #if DEBUG_INPUT
00114   fprintf (stderr, "input_free:  end\n");
00115 #endif
00116 }
```

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 = 0;
00072   input->simulator = input->evaluator = input->directory = input->name = NULL;
00073   input->experiment = NULL;
00074   input->variable = NULL;
00075 #if DEBUG_INPUT
00076   fprintf (stderr, "input_new:  end\n");
00077 #endif
00078 }
```

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

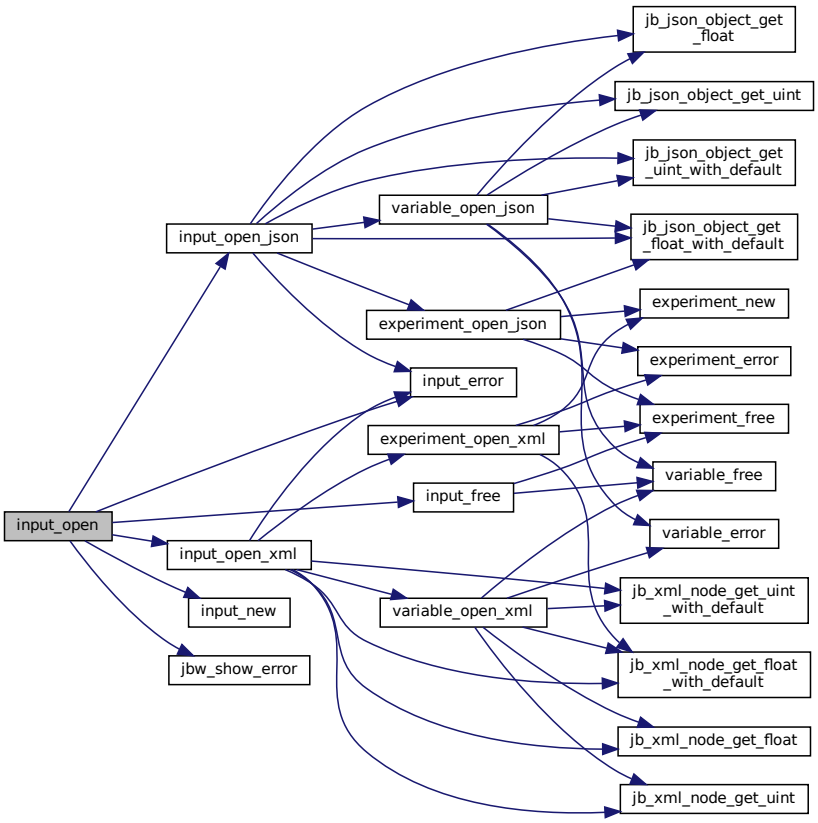
filename	Input data file name.

Definition at line 963 of file input.c.

```
00964 {
00965   xmlDoc *doc;
00966   JsonParser *parser;
00967
00968 #if DEBUG_INPUT
00969   fprintf (stderr, "input_open:  start\n");
00970 #endif
00971
00972   // Resetting input data
00973   input_new ();
00974
00975   // Opening input file
00976 #if DEBUG_INPUT
00977   fprintf (stderr, "input_open:  opening the input file %s\n", filename);
00978   fprintf (stderr, "input_open:  trying XML format\n");
00979 #endif
00980   doc = xmlParseFile (filename);
00981   if (!doc)
00982     {
00983 #if DEBUG_INPUT
00984       fprintf (stderr, "input_open:  trying JSON format\n");
00985 #endif
00986       parser = json_parser_new ();
00987       if (!json_parser_load_from_file (parser, filename, NULL))
00988         {
00989           input_error (_("Unable to parse the input file"));
00990           goto exit_on_error;
00991         }
00992       if (!input_open_json (parser))
```

```
00993         goto exit_on_error;
00994       }
00995     else if (!input_open_xml (doc))
00996       goto exit_on_error;
00997
00998     // Getting the working directory
00999     input->directory = g_path_get_dirname (filename);
01000     input->name = g_path_get_basename (filename);
01001
01002 #if DEBUG_INPUT
01003     fprintf (stderr, "input_open:  end\n");
01004 #endif
01005     return 1;
01006
01007 exit_on_error:
01008     jbw_show_error (error_message);
01009     g_free (error_message);
01010     input_free ();
01011 #if DEBUG_INPUT
01012     fprintf (stderr, "input_open:  end\n");
01013 #endif
01014     return 0;
01015 }
```

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

parser	JsonParser struct.

Definition at line 572 of file input.c.

```
00573 {
00574   JsonNode *node, *child;
00575   JsonObject *object;
00576   JsonArray *array;
00577   const char *buffer;
00578   int error_code;
00579   unsigned int i, n;
00580
00581 #if DEBUG_INPUT
00582   fprintf (stderr, "input_open_json:  start\n");
00583 #endif
00584
00585   // Resetting input data
00586   input->type = INPUT_TYPE_JSON;
00587
00588   // Getting the root node
00589 #if DEBUG_INPUT
00590   fprintf (stderr, "input_open_json:  getting the root node\n");
00591 #endif
00592   node = json_parser_get_root (parser);
00593   object = json_node_get_object (node);
00594
00595   // Getting result and variables file names
00596   if (!input->result)
00597     {
00598       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00599       if (!buffer)
00600         buffer = result_name;
00601       input->result = g_strdup (buffer);
00602     }
00603   else
00604     input->result = g_strdup (result_name);
00605   if (!input->variables)
00606     {
00607       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00608       if (!buffer)
00609         buffer = variables_name;
00610       input->variables = g_strdup (buffer);
00611     }
00612   else
00613     input->variables = g_strdup (variables_name);
00614
00615   // Opening simulator program name
00616   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00617   if (!buffer)
00618     {
00619       input_error (_("Bad simulator program"));
00620       goto exit_on_error;
00621     }
00622   input->simulator = g_strdup (buffer);
00623
00624   // Opening evaluator program name
00625   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00626   if (buffer)
00627     input->evaluator = g_strdup (buffer);
00628
00629   // Obtaining pseudo-random numbers generator seed
00630   input->seed
00631     = jb_json_object_get_uint_with_default (object, LABEL_SEED,
00632                                             &error_code, DEFAULT_RANDOM_SEED);
00633   if (!error_code)
00634     {
00635       input_error (_("Bad pseudo-random numbers generator seed"));
00636       goto exit_on_error;
00637     }
00638
00639   // Opening algorithm
00640   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00641   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00642     {
00643       input->algorithm = ALGORITHM_MONTE_CARLO;
00644
```

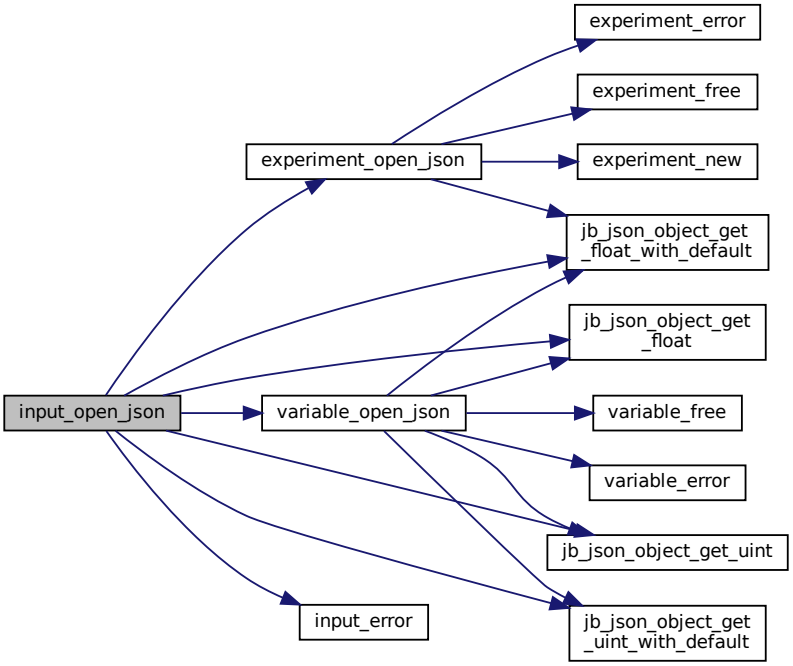
```
00645       // Obtaining simulations number
00646       input->nsimulations
00647         = jb_json_object_get_uint (object, LABEL_NSIMULATIONS, &error_code);
00648       if (!error_code)
00649         {
00650           input_error (_("Bad simulations number"));
00651           goto exit_on_error;
00652         }
00653     }
00654   else if (!strcmp (buffer, LABEL_SWEEP))
00655     input->algorithm = ALGORITHM_SWEEP;
00656   else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00657     input->algorithm = ALGORITHM_ORTHOGONAL;
00658   else if (!strcmp (buffer, LABEL_GENETIC))
00659     {
00660       input->algorithm = ALGORITHM_GENETIC;
00661
00662       // Obtaining population
00663       if (json_object_get_member (object, LABEL_NPOPULATION))
00664         {
00665           input->nsimulations
00666             = jb_json_object_get_uint (object, LABEL_NPOPULATION, &error_code);
00667           if (!error_code || input->nsimulations < 3)
00668             {
00669               input_error (_("Invalid population number"));
00670               goto exit_on_error;
00671             }
00672         }
00673       else
00674         {
00675           input_error (_("No population number"));
00676           goto exit_on_error;
00677         }
00678
00679       // Obtaining generations
00680       if (json_object_get_member (object, LABEL_NGENERATIONS))
00681         {
00682           input->niterations
00683             = jb_json_object_get_uint_with_default (object, LABEL_NGENERATIONS,
00684                                                     &error_code, 1);
00685           if (!error_code || !input->niterations)
00686             {
00687               input_error (_("Invalid generations number"));
00688               goto exit_on_error;
00689             }
00690         }
00691       else
00692         {
00693           input_error (_("No generations number"));
00694           goto exit_on_error;
00695         }
00696
00697       // Obtaining mutation probability
00698       if (json_object_get_member (object, LABEL_MUTATION))
00699         {
00700           input->mutation_ratio
00701             = jb_json_object_get_float (object, LABEL_MUTATION, &error_code);
00702           if (!error_code || input->mutation_ratio < 0.
00703               || input->mutation_ratio >= 1.)
00704             {
00705               input_error (_("Invalid mutation probability"));
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         {
00711           input_error (_("No mutation probability"));
00712           goto exit_on_error;
00713         }
00714
00715       // Obtaining reproduction probability
00716       if (json_object_get_member (object, LABEL_REPRODUCTION))
00717         {
00718           input->reproduction_ratio
00719             = jb_json_object_get_float (object, LABEL_REPRODUCTION,
00720                                         &error_code);
00721           if (!error_code || input->reproduction_ratio < 0.
00722               || input->reproduction_ratio >= 1.0)
00723             {
00724               input_error (_("Invalid reproduction probability"));
00725               goto exit_on_error;
00726             }
00727         }
00728       else
00729         {
00730           input_error (_("No reproduction probability"));
00731           goto exit_on_error;
```

```
00732         }
00733
00734       // Obtaining adaptation probability
00735       if (json_object_get_member (object, LABEL_ADAPTATION))
00736         {
00737           input->adaptation_ratio
00738             = jb_json_object_get_float (object, LABEL_ADAPTATION, &error_code);
00739           if (!error_code || input->adaptation_ratio < 0.
00740               || input->adaptation_ratio >= 1.)
00741             {
00742               input_error (_("Invalid adaptation probability"));
00743               goto exit_on_error;
00744             }
00745         }
00746       else
00747         {
00748           input_error (_("No adaptation probability"));
00749           goto exit_on_error;
00750         }
00751
00752       // Checking survivals
00753       i = input->mutation_ratio * input->nsimulations;
00754       i += input->reproduction_ratio * input->nsimulations;
00755       i += input->adaptation_ratio * input->nsimulations;
00756       if (i > input->nsimulations - 2)
00757         {
00758           input_error
00759             (_("No enough survival entities to reproduce the population"));
00760           goto exit_on_error;
00761         }
00762     }
00763   else
00764     {
00765       input_error (_("Unknown algorithm"));
00766       goto exit_on_error;
00767     }
00768
00769   if (input->algorithm == ALGORITHM_MONTE_CARLO
00770       || input->algorithm == ALGORITHM_SWEEP
00771       || input->algorithm == ALGORITHM_ORTHOGONAL)
00772     {
00773
00774       // Obtaining iterations number
00775       input->niterations
00776         = jb_json_object_get_uint (object, LABEL_NITERATIONS, &error_code);
00777       if (!error_code || !input->niterations)
00778         {
00779           input_error (_("Bad iterations number"));
00780           goto exit_on_error;
00781         }
00782
00783       // Obtaining best number
00784       input->nbest
00785         = jb_json_object_get_uint_with_default (object, LABEL_NBEST,
00786                                                 &error_code, 1);
00787       if (!error_code || !input->nbest)
00788         {
00789           input_error (_("Invalid best number"));
00790           goto exit_on_error;
00791         }
00792
00793       // Obtaining tolerance
00794       input->tolerance
00795         = jb_json_object_get_float_with_default (object, LABEL_TOLERANCE,
00796                                                  &error_code, 0.);
00797       if (!error_code || input->tolerance < 0.)
00798         {
00799           input_error (_("Invalid tolerance"));
00800           goto exit_on_error;
00801         }
00802
00803       // Getting hill climbing method parameters
00804       if (json_object_get_member (object, LABEL_NSTEPS))
00805         {
00806           input->nsteps
00807             = jb_json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00808           if (!error_code)
00809             {
00810               input_error (_("Invalid steps number"));
00811               goto exit_on_error;
00812             }
00813           buffer = json_object_get_string_member (object, LABEL_CLIMBING);
00814           if (!strcmp (buffer, LABEL_COORDINATES))
00815             input->climbing = CLIMBING_METHOD_COORDINATES;
00816           else if (!strcmp (buffer, LABEL_RANDOM))
00817             {
00818               input->climbing = CLIMBING_METHOD_RANDOM;
```

```
00819               input->nestimates
00820                 = jb_json_object_get_uint (object, LABEL_NESTIMATES,
00821                                            &error_code);
00822               if (!error_code || !input->nestimates)
00823                 {
00824                   input_error (_("Invalid estimates number"));
00825                   goto exit_on_error;
00826                 }
00827             }
00828           else
00829             {
00830               input_error (_("Unknown method to estimate the hill climbing"));
00831               goto exit_on_error;
00832             }
00833           input->relaxation
00834             = jb_json_object_get_float_with_default (object, LABEL_RELAXATION,
00835                                                      &error_code,
00836                                                      DEFAULT_RELAXATION);
00837           if (!error_code || input->relaxation < 0.  || input->relaxation > 2.)
00838             {
00839               input_error (_("Invalid relaxation parameter"));
00840               goto exit_on_error;
00841             }
00842         }
00843       else
00844         input->nsteps = 0;
00845     }
00846   // Obtaining the threshold
00847   input->threshold
00848     = jb_json_object_get_float_with_default (object, LABEL_THRESHOLD,
00849                                              &error_code, 0.);
00850
00851   if (!error_code)
00852     {
00853       input_error (_("Invalid threshold"));
00854       goto exit_on_error;
00855     }
00856
00857   // Reading the experimental data
00858   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00859   n = json_array_get_length (array);
00860   input->experiment = (Experiment *) g_malloc (n * sizeof (Experiment));
00861   for (i = 0; i < n; ++i)
00862     {
00863 #if DEBUG_INPUT
00864       fprintf (stderr, "input_open_json:  nexperiments=%u\n",
00865                input->nexperiments);
00866 #endif
00867       child = json_array_get_element (array, i);
00868       if (!input->nexperiments)
00869         {
00870           if (!experiment_open_json (input->experiment, child, 0))
00871             goto exit_on_error;
00872         }
00873       else
00874         {
00875           if (!experiment_open_json (input->experiment + input->nexperiments,
00876                                      child, input->experiment->ninputs))
00877             goto exit_on_error;
00878         }
00879       ++input->nexperiments;
00880 #if DEBUG_INPUT
00881       fprintf (stderr, "input_open_json:  nexperiments=%u\n",
00882                input->nexperiments);
00883 #endif
00884     }
00885   if (!input->nexperiments)
00886     {
00887       input_error (_("No optimization experiments"));
00888       goto exit_on_error;
00889     }
00890
00891   // Reading the variables data
00892   array = json_object_get_array_member (object, LABEL_VARIABLES);
00893   n = json_array_get_length (array);
00894   input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00895   for (i = 0; i < n; ++i)
00896     {
00897 #if DEBUG_INPUT
00898       fprintf (stderr, "input_open_json:  nvariables=%u\n", input->nvariables);
00899 #endif
00900       child = json_array_get_element (array, i);
00901       if (!variable_open_json (input->variable + input->nvariables, child,
00902                                input->algorithm, input->nsteps))
00903         goto exit_on_error;
00904       ++input->nvariables;
00905     }
```

```
00906   if (!input->nvariables)
00907     {
00908       input_error (_("No optimization variables"));
00909       goto exit_on_error;
00910     }
00911
00912   // Obtaining the error norm
00913   if (json_object_get_member (object, LABEL_NORM))
00914     {
00915       buffer = json_object_get_string_member (object, LABEL_NORM);
00916       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00917         input->norm = ERROR_NORM_EUCLIDIAN;
00918       else if (!strcmp (buffer, LABEL_MAXIMUM))
00919         input->norm = ERROR_NORM_MAXIMUM;
00920       else if (!strcmp (buffer, LABEL_P))
00921         {
00922           input->norm = ERROR_NORM_P;
00923           input->p = jb_json_object_get_float (object, LABEL_P, &error_code);
00924           if (!error_code)
00925             {
00926               input_error (_("Bad P parameter"));
00927               goto exit_on_error;
00928             }
00929         }
00930       else if (!strcmp (buffer, LABEL_TAXICAB))
00931         input->norm = ERROR_NORM_TAXICAB;
00932       else
00933         {
00934           input_error (_("Unknown error norm"));
00935           goto exit_on_error;
00936         }
00937     }
00938   else
00939     input->norm = ERROR_NORM_EUCLIDIAN;
00940
00941   // Closing the JSON document
00942   g_object_unref (parser);
00943
00944 #if DEBUG_INPUT
00945   fprintf (stderr, "input_open_json:  end\n");
00946 #endif
00947   return 1;
00948
00949 exit_on_error:
00950   g_object_unref (parser);
00951 #if DEBUG_INPUT
00952   fprintf (stderr, "input_open_json:  end\n");
00953 #endif
00954   return 0;
00955 }
```

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

doc	xmlDoc struct.

Definition at line 135 of file input.c.

```
00136 {
00137   char buffer2[64];
00138   xmlNode *node, *child;
00139   xmlChar *buffer;
00140   int error_code;
00141   unsigned int i;
00142
00143 #if DEBUG_INPUT
00144   fprintf (stderr, "input_open_xml:  start\n");
```

```
00145 #endif
00146
00147   // Resetting input data
00148   buffer = NULL;
00149   input->type = INPUT_TYPE_XML;
00150
00151   // Getting the root node
00152 #if DEBUG_INPUT
00153   fprintf (stderr, "input_open_xml:  getting the root node\n");
00154 #endif
00155   node = xmlDocGetRootElement (doc);
00156   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00157     {
00158       input_error (_("Bad root XML node"));
00159       goto exit_on_error;
00160     }
00161
00162   // Getting result and variables file names
00163   if (!input->result)
00164     {
00165       input->result =
00166         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00167       if (!input->result)
00168         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00169     }
00170 #if DEBUG_INPUT
00171   fprintf (stderr, "input_open_xml:  result file=%s\n", input->result);
00172 #endif
00173   if (!input->variables)
00174     {
00175       input->variables =
00176         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00177       if (!input->variables)
00178         input->variables =
00179           (char *) xmlStrdup ((const xmlChar *) variables_name);
00180     }
00181 #if DEBUG_INPUT
00182   fprintf (stderr, "input_open_xml:  variables file=%s\n", input->variables);
00183 #endif
00184
00185   // Opening simulator program name
00186   input->simulator =
00187     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00188   if (!input->simulator)
00189     {
00190       input_error (_("Bad simulator program"));
00191       goto exit_on_error;
00192     }
00193
00194   // Opening evaluator program name
00195   input->evaluator =
00196     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00197
00198   // Obtaining pseudo-random numbers generator seed
00199   input->seed
00200     = jb_xml_node_get_uint_with_default (node, (const xmlChar *) LABEL_SEED,
00201                                          &error_code, DEFAULT_RANDOM_SEED);
00202   if (!error_code)
00203     {
00204       input_error (_("Bad pseudo-random numbers generator seed"));
00205       goto exit_on_error;
00206     }
00207
00208   // Opening algorithm
00209   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00210   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00211     {
00212       input->algorithm = ALGORITHM_MONTE_CARLO;
00213
00214       // Obtaining simulations number
00215       input->nsimulations
00216         = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSIMULATIONS,
00217                                 &error_code);
00218       if (!error_code)
00219         {
00220           input_error (_("Bad simulations number"));
00221           goto exit_on_error;
00222         }
00223     }
00224   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00225     input->algorithm = ALGORITHM_SWEEP;
00226   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00227     input->algorithm = ALGORITHM_ORTHOGONAL;
00228   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00229     {
00230       input->algorithm = ALGORITHM_GENETIC;
00231
```

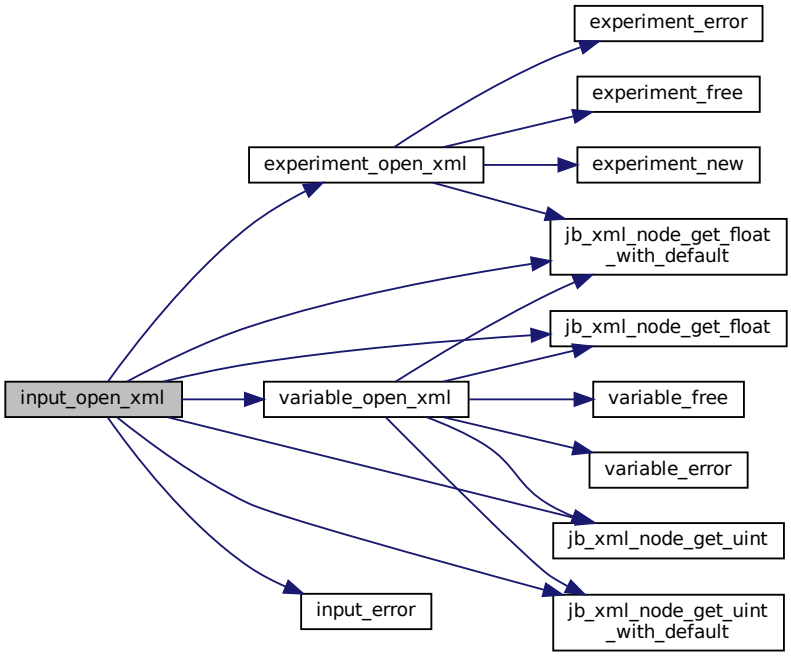
```
00232       // Obtaining population
00233       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00234         {
00235           input->nsimulations
00236             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00237                                     &error_code);
00238           if (!error_code || input->nsimulations < 3)
00239             {
00240               input_error (_("Invalid population number"));
00241               goto exit_on_error;
00242             }
00243         }
00244       else
00245         {
00246           input_error (_("No population number"));
00247           goto exit_on_error;
00248         }
00249
00250       // Obtaining generations
00251       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00252         {
00253           input->niterations
00254             = jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00255                                     &error_code);
00256           if (!error_code || !input->niterations)
00257             {
00258               input_error (_("Invalid generations number"));
00259               goto exit_on_error;
00260             }
00261         }
00262       else
00263         {
00264           input_error (_("No generations number"));
00265           goto exit_on_error;
00266         }
00267
00268       // Obtaining mutation probability
00269       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00270         {
00271           input->mutation_ratio
00272             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00273                                      &error_code);
00274           if (!error_code || input->mutation_ratio < 0.
00275               || input->mutation_ratio >= 1.)
00276             {
00277               input_error (_("Invalid mutation probability"));
00278               goto exit_on_error;
00279             }
00280         }
00281       else
00282         {
00283           input_error (_("No mutation probability"));
00284           goto exit_on_error;
00285         }
00286
00287       // Obtaining reproduction probability
00288       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00289         {
00290           input->reproduction_ratio
00291             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00292                                      &error_code);
00293           if (!error_code || input->reproduction_ratio < 0.
00294               || input->reproduction_ratio >= 1.0)
00295             {
00296               input_error (_("Invalid reproduction probability"));
00297               goto exit_on_error;
00298             }
00299         }
00300       else
00301         {
00302           input_error (_("No reproduction probability"));
00303           goto exit_on_error;
00304         }
00305
00306       // Obtaining adaptation probability
00307       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00308         {
00309           input->adaptation_ratio
00310             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00311                                      &error_code);
00312           if (!error_code || input->adaptation_ratio < 0.
00313               || input->adaptation_ratio >= 1.)
00314             {
00315               input_error (_("Invalid adaptation probability"));
00316               goto exit_on_error;
00317             }
00318         }
```

```
00319       else
00320         {
00321           input_error (_("No adaptation probability"));
00322           goto exit_on_error;
00323         }
00324
00325       // Checking survivals
00326       i = input->mutation_ratio * input->nsimulations;
00327       i += input->reproduction_ratio * input->nsimulations;
00328       i += input->adaptation_ratio * input->nsimulations;
00329       if (i > input->nsimulations - 2)
00330         {
00331           input_error
00332             (_("No enough survival entities to reproduce the population"));
00333           goto exit_on_error;
00334         }
00335     }
00336   else
00337     {
00338       input_error (_("Unknown algorithm"));
00339       goto exit_on_error;
00340     }
00341   xmlFree (buffer);
00342   buffer = NULL;
00343
00344   if (input->algorithm == ALGORITHM_MONTE_CARLO
00345       || input->algorithm == ALGORITHM_SWEEP
00346       || input->algorithm == ALGORITHM_ORTHOGONAL)
00347     {
00348
00349       // Obtaining iterations number
00350       input->niterations = jb_xml_node_get_uint_with_default
00351         (node, (const xmlChar *) LABEL_NITERATIONS, &error_code, 1);
00352       if (!error_code || !input->niterations)
00353         {
00354           input_error (_("Bad iterations number"));
00355           goto exit_on_error;
00356         }
00357
00358       // Obtaining best number
00359       input->nbest
00360         = jb_xml_node_get_uint_with_default (node,
00361                                              (const xmlChar *) LABEL_NBEST,
00362                                              &error_code, 1);
00363       if (!error_code || !input->nbest)
00364         {
00365           input_error (_("Invalid best number"));
00366           goto exit_on_error;
00367         }
00368
00369       // Obtaining tolerance
00370       input->tolerance
00371         = jb_xml_node_get_float_with_default (node,
00372                                               (const xmlChar *) LABEL_TOLERANCE,
00373                                               &error_code, 0.);
00374       if (!error_code || input->tolerance < 0.)
00375         {
00376           input_error (_("Invalid tolerance"));
00377           goto exit_on_error;
00378         }
00379
00380       // Getting hill climbing method parameters
00381       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00382         {
00383           input->nsteps =
00384             jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00385                                   &error_code);
00386           if (!error_code)
00387             {
00388               input_error (_("Invalid steps number"));
00389               goto exit_on_error;
00390             }
00391 #if DEBUG_INPUT
00392           fprintf (stderr, "input_open_xml:  nsteps=%u\n", input->nsteps);
00393 #endif
00394           buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00395           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00396             input->climbing = CLIMBING_METHOD_COORDINATES;
00397           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00398             {
00399               input->climbing = CLIMBING_METHOD_RANDOM;
00400               input->nestimates
00401                 = jb_xml_node_get_uint (node,
00402                                         (const xmlChar *) LABEL_NESTIMATES,
00403                                         &error_code);
00404               if (!error_code || !input->nestimates)
00405                 {
```

```
00406                   input_error (_("Invalid estimates number"));
00407                   goto exit_on_error;
00408                 }
00409             }
00410           else
00411             {
00412               input_error (_("Unknown method to estimate the hill climbing"));
00413               goto exit_on_error;
00414             }
00415           xmlFree (buffer);
00416           buffer = NULL;
00417           input->relaxation
00418             = jb_xml_node_get_float_with_default (node,
00419                                                   (const xmlChar *)
00420                                                   LABEL_RELAXATION,
00421                                                   &error_code,
00422                                                   DEFAULT_RELAXATION);
00423           if (!error_code || input->relaxation < 0.  || input->relaxation > 2.)
00424             {
00425               input_error (_("Invalid relaxation parameter"));
00426               goto exit_on_error;
00427             }
00428         }
00429       else
00430         input->nsteps = 0;
00431     }
00432   // Obtaining the threshold
00433   input->threshold =
00434     jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_THRESHOLD,
00435                                         &error_code, 0.);
00436   if (!error_code)
00437     {
00438       input_error (_("Invalid threshold"));
00439       goto exit_on_error;
00440     }
00441
00442   // Reading the experimental data
00443   for (child = node->children; child; child = child->next)
00444     {
00445       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00446         break;
00447 #if DEBUG_INPUT
00448       fprintf (stderr, "input_open_xml:  nexperiments=%u\n",
00449                input->nexperiments);
00450 #endif
00451       input->experiment = (Experiment *)
00452         g_realloc (input->experiment,
00453                    (1 + input->nexperiments) * sizeof (Experiment));
00454       if (!input->nexperiments)
00455         {
00456           if (!experiment_open_xml (input->experiment, child, 0))
00457             goto exit_on_error;
00458         }
00459       else
00460         {
00461           if (!experiment_open_xml (input->experiment + input->nexperiments,
00462                                     child, input->experiment->ninputs))
00463             goto exit_on_error;
00464         }
00465       ++input->nexperiments;
00466 #if DEBUG_INPUT
00467       fprintf (stderr, "input_open_xml:  nexperiments=%u\n",
00468                input->nexperiments);
00469 #endif
00470     }
00471   if (!input->nexperiments)
00472     {
00473       input_error (_("No optimization experiments"));
00474       goto exit_on_error;
00475     }
00476   buffer = NULL;
00477
00478   // Reading the variables data
00479   if (input->algorithm == ALGORITHM_SWEEP
00480       || input->algorithm == ALGORITHM_ORTHOGONAL)
00481     input->nsimulations = 1;
00482   for (; child; child = child->next)
00483     {
00484 #if DEBUG_INPUT
00485       fprintf (stderr, "input_open_xml:  nvariables=%u\n", input->nvariables);
00486 #endif
00487       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00488         {
00489           snprintf (buffer2, 64, "%s %u:  %s",
00490                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00491           input_error (buffer2);
00492           goto exit_on_error;
```

```
00493         }
00494       input->variable = (Variable *)
00495         g_realloc (input->variable,
00496                    (1 + input->nvariables) * sizeof (Variable));
00497       if (!variable_open_xml (input->variable + input->nvariables, child,
00498                               input->algorithm, input->nsteps))
00499         goto exit_on_error;
00500       if (input->algorithm == ALGORITHM_SWEEP
00501           || input->algorithm == ALGORITHM_ORTHOGONAL)
00502         input->nsimulations *= input->variable[input->nvariables].nsweeps;
00503       ++input->nvariables;
00504     }
00505   if (!input->nvariables)
00506     {
00507       input_error (_("No optimization variables"));
00508       goto exit_on_error;
00509     }
00510   if (input->nbest > input->nsimulations)
00511     {
00512       input_error (_("Best number higher than simulations number"));
00513       goto exit_on_error;
00514     }
00515   buffer = NULL;
00516
00517   // Obtaining the error norm
00518   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00519     {
00520       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00521       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00522         input->norm = ERROR_NORM_EUCLIDIAN;
00523       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00524         input->norm = ERROR_NORM_MAXIMUM;
00525       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00526         {
00527           input->norm = ERROR_NORM_P;
00528           input->p
00529             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_P,
00530                                      &error_code);
00531           if (!error_code)
00532             {
00533               input_error (_("Bad P parameter"));
00534               goto exit_on_error;
00535             }
00536         }
00537       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00538         input->norm = ERROR_NORM_TAXICAB;
00539       else
00540         {
00541           input_error (_("Unknown error norm"));
00542           goto exit_on_error;
00543         }
00544       xmlFree (buffer);
00545     }
00546   else
00547     input->norm = ERROR_NORM_EUCLIDIAN;
00548
00549   // Closing the XML document
00550   xmlFreeDoc (doc);
00551
00552 #if DEBUG_INPUT
00553   fprintf (stderr, "input_open_xml:  end\n");
00554 #endif
00555   return 1;
00556
00557 exit_on_error:
00558   xmlFree (buffer);
00559   xmlFreeDoc (doc);
00560 #if DEBUG_INPUT
00561   fprintf (stderr, "input_open_xml:  end\n");
00562 #endif
00563   return 0;
00564 }
```

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 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
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <string.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #include <glib.h>
00045 #include <glib/gstdio.h>
00046 #include <json-glib/json-glib.h>
00047 #include "jb/src/jb_xml.h"
00048 #include "jb/src/jb_json.h"
00049 #include "jb/src/jb_win.h"
00050 #include "tools.h"
00051 #include "experiment.h"
00052 #include "variable.h"
00053 #include "input.h"
00054
00055 #define DEBUG_INPUT 0
00056
00057 Input input[1];
00058
00059 const char *result_name = "result";
00060 const char *variables_name = "variables";
00061
00065 void
00066 input_new ()
00067 {
00068 #if DEBUG_INPUT
00069   fprintf (stderr, "input_new:  start\n");
00070 #endif
00071   input->nvariables = input->nexperiments = input->nsteps = 0;
00072   input->simulator = input->evaluator = input->directory = input->name = NULL;
00073   input->experiment = NULL;
00074   input->variable = NULL;
00075 #if DEBUG_INPUT
```

```
00076   fprintf (stderr, "input_new:  end\n");
00077 #endif
00078 }
00079
00083 void
00084 input_free ()
00085 {
00086   unsigned int i;
00087 #if DEBUG_INPUT
00088   fprintf (stderr, "input_free:  start\n");
00089 #endif
00090   g_free (input->name);
00091   g_free (input->directory);
00092   for (i = 0; i < input->nexperiments; ++i)
00093     experiment_free (input->experiment + i, input->type);
00094   for (i = 0; i < input->nvariables; ++i)
00095     variable_free (input->variable + i, input->type);
00096   g_free (input->experiment);
00097   g_free (input->variable);
00098   if (input->type == INPUT_TYPE_XML)
00099     {
00100       xmlFree (input->evaluator);
00101       xmlFree (input->simulator);
00102       xmlFree (input->result);
00103       xmlFree (input->variables);
00104     }
00105   else
00106     {
00107       g_free (input->evaluator);
00108       g_free (input->simulator);
00109       g_free (input->result);
00110       g_free (input->variables);
00111     }
00112   input->nexperiments = input->nvariables = input->nsteps = 0;
00113 #if DEBUG_INPUT
00114   fprintf (stderr, "input_free:  end\n");
00115 #endif
00116 }
00117
00121 static void
00122 input_error (char *message)
00123 {
00124   char buffer[64];
00125   snprintf (buffer, 64, "%s:  %s\n", _("Input"), message);
00126   error_message = g_strdup (buffer);
00127 }
00128
00134 static inline int
00135 input_open_xml (xmlDoc * doc)
00136 {
00137   char buffer2[64];
00138   xmlNode *node, *child;
00139   xmlChar *buffer;
00140   int error_code;
00141   unsigned int i;
00142
00143 #if DEBUG_INPUT
00144   fprintf (stderr, "input_open_xml:  start\n");
00145 #endif
00146
00147   // Resetting input data
00148   buffer = NULL;
00149   input->type = INPUT_TYPE_XML;
00150
00151   // Getting the root node
00152 #if DEBUG_INPUT
00153   fprintf (stderr, "input_open_xml:  getting the root node\n");
00154 #endif
00155   node = xmlDocGetRootElement (doc);
00156   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00157     {
00158       input_error (_("Bad root XML node"));
00159       goto exit_on_error;
00160     }
00161
00162   // Getting result and variables file names
00163   if (!input->result)
00164     {
00165       input->result =
00166         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00167       if (!input->result)
00168         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00169     }
00170 #if DEBUG_INPUT
00171   fprintf (stderr, "input_open_xml:  result file=%s\n", input->result);
00172 #endif
00173   if (!input->variables)
```

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

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

```
00348
00349       // Obtaining iterations number
00350       input->niterations = jb_xml_node_get_uint_with_default
00351         (node, (const xmlChar *) LABEL_NITERATIONS, &error_code, 1);
00352       if (!error_code || !input->niterations)
00353         {
00354           input_error (_("Bad iterations number"));
00355           goto exit_on_error;
00356         }
00357
00358       // Obtaining best number
00359       input->nbest
00360         = jb_xml_node_get_uint_with_default (node,
00361                                              (const xmlChar *) LABEL_NBEST,
00362                                              &error_code, 1);
00363       if (!error_code || !input->nbest)
00364         {
00365           input_error (_("Invalid best number"));
00366           goto exit_on_error;
00367         }
00368
00369       // Obtaining tolerance
00370       input->tolerance
00371         = jb_xml_node_get_float_with_default (node,
00372                                               (const xmlChar *) LABEL_TOLERANCE,
00373                                               &error_code, 0.);
00374       if (!error_code || input->tolerance < 0.)
00375         {
00376           input_error (_("Invalid tolerance"));
00377           goto exit_on_error;
00378         }
00379
00380       // Getting hill climbing method parameters
00381       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00382         {
00383           input->nsteps =
00384             jb_xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00385                                   &error_code);
00386           if (!error_code)
00387             {
00388               input_error (_("Invalid steps number"));
00389               goto exit_on_error;
00390             }
00391 #if DEBUG_INPUT
00392           fprintf (stderr, "input_open_xml:  nsteps=%u\n", input->nsteps);
00393 #endif
00394           buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00395           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00396             input->climbing = CLIMBING_METHOD_COORDINATES;
00397           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00398             {
00399               input->climbing = CLIMBING_METHOD_RANDOM;
00400               input->nestimates
00401                 = jb_xml_node_get_uint (node,
00402                                         (const xmlChar *) LABEL_NESTIMATES,
00403                                         &error_code);
00404               if (!error_code || !input->nestimates)
00405                 {
00406                   input_error (_("Invalid estimates number"));
00407                   goto exit_on_error;
00408                 }
00409             }
00410           else
00411             {
00412               input_error (_("Unknown method to estimate the hill climbing"));
00413               goto exit_on_error;
00414             }
00415           xmlFree (buffer);
00416           buffer = NULL;
00417           input->relaxation
00418             = jb_xml_node_get_float_with_default (node,
00419                                                   (const xmlChar *)
00420                                                   LABEL_RELAXATION,
00421                                                   &error_code,
00422                                                   DEFAULT_RELAXATION);
00423           if (!error_code || input->relaxation < 0.  || input->relaxation > 2.)
00424             {
00425               input_error (_("Invalid relaxation parameter"));
00426               goto exit_on_error;
00427             }
00428         }
00429       else
00430         input->nsteps = 0;
00431     }
00432   // Obtaining the threshold
00433   input->threshold =
00434     jb_xml_node_get_float_with_default (node, (const xmlChar *) LABEL_THRESHOLD,
```

```
00435                                       &error_code, 0.);
00436   if (!error_code)
00437     {
00438       input_error (_("Invalid threshold"));
00439       goto exit_on_error;
00440     }
00441
00442   // Reading the experimental data
00443   for (child = node->children; child; child = child->next)
00444     {
00445       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00446         break;
00447 #if DEBUG_INPUT
00448       fprintf (stderr, "input_open_xml:  nexperiments=%u\n",
00449                input->nexperiments);
00450 #endif
00451       input->experiment = (Experiment *)
00452         g_realloc (input->experiment,
00453                    (1 + input->nexperiments) * sizeof (Experiment));
00454       if (!input->nexperiments)
00455         {
00456           if (!experiment_open_xml (input->experiment, child, 0))
00457             goto exit_on_error;
00458         }
00459       else
00460         {
00461           if (!experiment_open_xml (input->experiment + input->nexperiments,
00462                                     child, input->experiment->ninputs))
00463             goto exit_on_error;
00464         }
00465       ++input->nexperiments;
00466 #if DEBUG_INPUT
00467       fprintf (stderr, "input_open_xml:  nexperiments=%u\n",
00468                input->nexperiments);
00469 #endif
00470     }
00471   if (!input->nexperiments)
00472     {
00473       input_error (_("No optimization experiments"));
00474       goto exit_on_error;
00475     }
00476   buffer = NULL;
00477
00478   // Reading the variables data
00479   if (input->algorithm == ALGORITHM_SWEEP
00480       || input->algorithm == ALGORITHM_ORTHOGONAL)
00481     input->nsimulations = 1;
00482   for (; child; child = child->next)
00483     {
00484 #if DEBUG_INPUT
00485       fprintf (stderr, "input_open_xml:  nvariables=%u\n", input->nvariables);
00486 #endif
00487       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00488         {
00489           snprintf (buffer2, 64, "%s %u:  %s",
00490                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00491           input_error (buffer2);
00492           goto exit_on_error;
00493         }
00494       input->variable = (Variable *)
00495         g_realloc (input->variable,
00496                    (1 + input->nvariables) * sizeof (Variable));
00497       if (!variable_open_xml (input->variable + input->nvariables, child,
00498                               input->algorithm, input->nsteps))
00499         goto exit_on_error;
00500       if (input->algorithm == ALGORITHM_SWEEP
00501           || input->algorithm == ALGORITHM_ORTHOGONAL)
00502         input->nsimulations *= input->variable[input->nvariables].nsweeps;
00503       ++input->nvariables;
00504     }
00505   if (!input->nvariables)
00506     {
00507       input_error (_("No optimization variables"));
00508       goto exit_on_error;
00509     }
00510   if (input->nbest > input->nsimulations)
00511     {
00512       input_error (_("Best number higher than simulations number"));
00513       goto exit_on_error;
00514     }
00515   buffer = NULL;
00516
00517   // Obtaining the error norm
00518   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00519     {
00520       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00521       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
```

```
00522         input->norm = ERROR_NORM_EUCLIDIAN;
00523       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00524         input->norm = ERROR_NORM_MAXIMUM;
00525       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00526         {
00527           input->norm = ERROR_NORM_P;
00528           input->p
00529             = jb_xml_node_get_float (node, (const xmlChar *) LABEL_P,
00530                                      &error_code);
00531           if (!error_code)
00532             {
00533               input_error (_("Bad P parameter"));
00534               goto exit_on_error;
00535             }
00536         }
00537       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00538         input->norm = ERROR_NORM_TAXICAB;
00539       else
00540         {
00541           input_error (_("Unknown error norm"));
00542           goto exit_on_error;
00543         }
00544       xmlFree (buffer);
00545     }
00546   else
00547     input->norm = ERROR_NORM_EUCLIDIAN;
00548
00549   // Closing the XML document
00550   xmlFreeDoc (doc);
00551
00552 #if DEBUG_INPUT
00553   fprintf (stderr, "input_open_xml:  end\n");
00554 #endif
00555   return 1;
00556
00557 exit_on_error:
00558   xmlFree (buffer);
00559   xmlFreeDoc (doc);
00560 #if DEBUG_INPUT
00561   fprintf (stderr, "input_open_xml:  end\n");
00562 #endif
00563   return 0;
00564 }
00565
00571 static inline int
00572 input_open_json (JsonParser * parser)
00573 {
00574   JsonNode *node, *child;
00575   JsonObject *object;
00576   JsonArray *array;
00577   const char *buffer;
00578   int error_code;
00579   unsigned int i, n;
00580
00581 #if DEBUG_INPUT
00582   fprintf (stderr, "input_open_json:  start\n");
00583 #endif
00584
00585   // Resetting input data
00586   input->type = INPUT_TYPE_JSON;
00587
00588   // Getting the root node
00589 #if DEBUG_INPUT
00590   fprintf (stderr, "input_open_json:  getting the root node\n");
00591 #endif
00592   node = json_parser_get_root (parser);
00593   object = json_node_get_object (node);
00594
00595   // Getting result and variables file names
00596   if (!input->result)
00597     {
00598       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00599       if (!buffer)
00600         buffer = result_name;
00601       input->result = g_strdup (buffer);
00602     }
00603   else
00604     input->result = g_strdup (result_name);
00605   if (!input->variables)
00606     {
00607       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00608       if (!buffer)
00609         buffer = variables_name;
00610       input->variables = g_strdup (buffer);
00611     }
00612   else
00613     input->variables = g_strdup (variables_name);
```

```
00614
00615   // Opening simulator program name
00616   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00617   if (!buffer)
00618     {
00619       input_error (_("Bad simulator program"));
00620       goto exit_on_error;
00621     }
00622   input->simulator = g_strdup (buffer);
00623
00624   // Opening evaluator program name
00625   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00626   if (buffer)
00627     input->evaluator = g_strdup (buffer);
00628
00629   // Obtaining pseudo-random numbers generator seed
00630   input->seed
00631     = jb_json_object_get_uint_with_default (object, LABEL_SEED,
00632                                             &error_code, DEFAULT_RANDOM_SEED);
00633   if (!error_code)
00634     {
00635       input_error (_("Bad pseudo-random numbers generator seed"));
00636       goto exit_on_error;
00637     }
00638
00639   // Opening algorithm
00640   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00641   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00642     {
00643       input->algorithm = ALGORITHM_MONTE_CARLO;
00644
00645       // Obtaining simulations number
00646       input->nsimulations
00647         = jb_json_object_get_uint (object, LABEL_NSIMULATIONS, &error_code);
00648       if (!error_code)
00649         {
00650           input_error (_("Bad simulations number"));
00651           goto exit_on_error;
00652         }
00653     }
00654   else if (!strcmp (buffer, LABEL_SWEEP))
00655     input->algorithm = ALGORITHM_SWEEP;
00656   else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00657     input->algorithm = ALGORITHM_ORTHOGONAL;
00658   else if (!strcmp (buffer, LABEL_GENETIC))
00659     {
00660       input->algorithm = ALGORITHM_GENETIC;
00661
00662       // Obtaining population
00663       if (json_object_get_member (object, LABEL_NPOPULATION))
00664         {
00665           input->nsimulations
00666             = jb_json_object_get_uint (object, LABEL_NPOPULATION, &error_code);
00667           if (!error_code || input->nsimulations < 3)
00668             {
00669               input_error (_("Invalid population number"));
00670               goto exit_on_error;
00671             }
00672         }
00673       else
00674         {
00675           input_error (_("No population number"));
00676           goto exit_on_error;
00677         }
00678
00679       // Obtaining generations
00680       if (json_object_get_member (object, LABEL_NGENERATIONS))
00681         {
00682           input->niterations
00683             = jb_json_object_get_uint_with_default (object, LABEL_NGENERATIONS,
00684                                                     &error_code, 1);
00685           if (!error_code || !input->niterations)
00686             {
00687               input_error (_("Invalid generations number"));
00688               goto exit_on_error;
00689             }
00690         }
00691       else
00692         {
00693           input_error (_("No generations number"));
00694           goto exit_on_error;
00695         }
00696
00697       // Obtaining mutation probability
00698       if (json_object_get_member (object, LABEL_MUTATION))
00699         {
00700           input->mutation_ratio
```

```
00701             = jb_json_object_get_float (object, LABEL_MUTATION, &error_code);
00702           if (!error_code || input->mutation_ratio < 0.
00703               || input->mutation_ratio >= 1.)
00704             {
00705               input_error (_("Invalid mutation probability"));
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         {
00711           input_error (_("No mutation probability"));
00712           goto exit_on_error;
00713         }
00714
00715       // Obtaining reproduction probability
00716       if (json_object_get_member (object, LABEL_REPRODUCTION))
00717         {
00718           input->reproduction_ratio
00719             = jb_json_object_get_float (object, LABEL_REPRODUCTION,
00720                                         &error_code);
00721           if (!error_code || input->reproduction_ratio < 0.
00722               || input->reproduction_ratio >= 1.0)
00723             {
00724               input_error (_("Invalid reproduction probability"));
00725               goto exit_on_error;
00726             }
00727         }
00728       else
00729         {
00730           input_error (_("No reproduction probability"));
00731           goto exit_on_error;
00732         }
00733
00734       // Obtaining adaptation probability
00735       if (json_object_get_member (object, LABEL_ADAPTATION))
00736         {
00737           input->adaptation_ratio
00738             = jb_json_object_get_float (object, LABEL_ADAPTATION, &error_code);
00739           if (!error_code || input->adaptation_ratio < 0.
00740               || input->adaptation_ratio >= 1.)
00741             {
00742               input_error (_("Invalid adaptation probability"));
00743               goto exit_on_error;
00744             }
00745         }
00746       else
00747         {
00748           input_error (_("No adaptation probability"));
00749           goto exit_on_error;
00750         }
00751
00752       // Checking survivals
00753       i = input->mutation_ratio * input->nsimulations;
00754       i += input->reproduction_ratio * input->nsimulations;
00755       i += input->adaptation_ratio * input->nsimulations;
00756       if (i > input->nsimulations - 2)
00757         {
00758           input_error
00759             (_("No enough survival entities to reproduce the population"));
00760           goto exit_on_error;
00761         }
00762     }
00763   else
00764     {
00765       input_error (_("Unknown algorithm"));
00766       goto exit_on_error;
00767     }
00768
00769   if (input->algorithm == ALGORITHM_MONTE_CARLO
00770       || input->algorithm == ALGORITHM_SWEEP
00771       || input->algorithm == ALGORITHM_ORTHOGONAL)
00772     {
00773
00774       // Obtaining iterations number
00775       input->niterations
00776         = jb_json_object_get_uint (object, LABEL_NITERATIONS, &error_code);
00777       if (!error_code || !input->niterations)
00778         {
00779           input_error (_("Bad iterations number"));
00780           goto exit_on_error;
00781         }
00782
00783       // Obtaining best number
00784       input->nbest
00785         = jb_json_object_get_uint_with_default (object, LABEL_NBEST,
00786                                                 &error_code, 1);
00787       if (!error_code || !input->nbest)
```

```
00788         {
00789           input_error (_("Invalid best number"));
00790           goto exit_on_error;
00791         }
00792
00793       // Obtaining tolerance
00794       input->tolerance
00795         = jb_json_object_get_float_with_default (object, LABEL_TOLERANCE,
00796                                                  &error_code, 0.);
00797       if (!error_code || input->tolerance < 0.)
00798         {
00799           input_error (_("Invalid tolerance"));
00800           goto exit_on_error;
00801         }
00802
00803       // Getting hill climbing method parameters
00804       if (json_object_get_member (object, LABEL_NSTEPS))
00805         {
00806           input->nsteps
00807             = jb_json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00808           if (!error_code)
00809             {
00810               input_error (_("Invalid steps number"));
00811               goto exit_on_error;
00812             }
00813           buffer = json_object_get_string_member (object, LABEL_CLIMBING);
00814           if (!strcmp (buffer, LABEL_COORDINATES))
00815             input->climbing = CLIMBING_METHOD_COORDINATES;
00816           else if (!strcmp (buffer, LABEL_RANDOM))
00817             {
00818               input->climbing = CLIMBING_METHOD_RANDOM;
00819               input->nestimates
00820                 = jb_json_object_get_uint (object, LABEL_NESTIMATES,
00821                                            &error_code);
00822               if (!error_code || !input->nestimates)
00823                 {
00824                   input_error (_("Invalid estimates number"));
00825                   goto exit_on_error;
00826                 }
00827             }
00828           else
00829             {
00830               input_error (_("Unknown method to estimate the hill climbing"));
00831               goto exit_on_error;
00832             }
00833           input->relaxation
00834             = jb_json_object_get_float_with_default (object, LABEL_RELAXATION,
00835                                                      &error_code,
00836                                                      DEFAULT_RELAXATION);
00837           if (!error_code || input->relaxation < 0.  || input->relaxation > 2.)
00838             {
00839               input_error (_("Invalid relaxation parameter"));
00840               goto exit_on_error;
00841             }
00842         }
00843       else
00844         input->nsteps = 0;
00845     }
00846   // Obtaining the threshold
00847   input->threshold
00848     = jb_json_object_get_float_with_default (object, LABEL_THRESHOLD,
00849                                              &error_code, 0.);
00850
00851   if (!error_code)
00852     {
00853       input_error (_("Invalid threshold"));
00854       goto exit_on_error;
00855     }
00856
00857   // Reading the experimental data
00858   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00859   n = json_array_get_length (array);
00860   input->experiment = (Experiment *) g_malloc (n * sizeof (Experiment));
00861   for (i = 0; i < n; ++i)
00862     {
00863 #if DEBUG_INPUT
00864       fprintf (stderr, "input_open_json:  nexperiments=%u\n",
00865                input->nexperiments);
00866 #endif
00867       child = json_array_get_element (array, i);
00868       if (!input->nexperiments)
00869         {
00870           if (!experiment_open_json (input->experiment, child, 0))
00871             goto exit_on_error;
00872         }
00873       else
00874         {
```

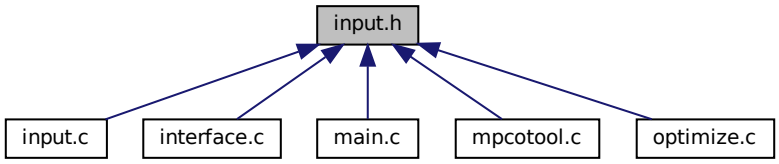
```
00875           if (!experiment_open_json (input->experiment + input->nexperiments,
00876                                      child, input->experiment->ninputs))
00877             goto exit_on_error;
00878         }
00879       ++input->nexperiments;
00880 #if DEBUG_INPUT
00881       fprintf (stderr, "input_open_json:  nexperiments=%u\n",
00882                input->nexperiments);
00883 #endif
00884     }
00885   if (!input->nexperiments)
00886     {
00887       input_error (_("No optimization experiments"));
00888       goto exit_on_error;
00889     }
00890
00891   // Reading the variables data
00892   array = json_object_get_array_member (object, LABEL_VARIABLES);
00893   n = json_array_get_length (array);
00894   input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00895   for (i = 0; i < n; ++i)
00896     {
00897 #if DEBUG_INPUT
00898       fprintf (stderr, "input_open_json:  nvariables=%u\n", input->nvariables);
00899 #endif
00900       child = json_array_get_element (array, i);
00901       if (!variable_open_json (input->variable + input->nvariables, child,
00902                                input->algorithm, input->nsteps))
00903         goto exit_on_error;
00904       ++input->nvariables;
00905     }
00906   if (!input->nvariables)
00907     {
00908       input_error (_("No optimization variables"));
00909       goto exit_on_error;
00910     }
00911
00912   // Obtaining the error norm
00913   if (json_object_get_member (object, LABEL_NORM))
00914     {
00915       buffer = json_object_get_string_member (object, LABEL_NORM);
00916       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00917         input->norm = ERROR_NORM_EUCLIDIAN;
00918       else if (!strcmp (buffer, LABEL_MAXIMUM))
00919         input->norm = ERROR_NORM_MAXIMUM;
00920       else if (!strcmp (buffer, LABEL_P))
00921         {
00922           input->norm = ERROR_NORM_P;
00923           input->p = jb_json_object_get_float (object, LABEL_P, &error_code);
00924           if (!error_code)
00925             {
00926               input_error (_("Bad P parameter"));
00927               goto exit_on_error;
00928             }
00929         }
00930       else if (!strcmp (buffer, LABEL_TAXICAB))
00931         input->norm = ERROR_NORM_TAXICAB;
00932       else
00933         {
00934           input_error (_("Unknown error norm"));
00935           goto exit_on_error;
00936         }
00937     }
00938   else
00939     input->norm = ERROR_NORM_EUCLIDIAN;
00940
00941   // Closing the JSON document
00942   g_object_unref (parser);
00943
00944 #if DEBUG_INPUT
00945   fprintf (stderr, "input_open_json:  end\n");
00946 #endif
00947   return 1;
00948
00949 exit_on_error:
00950   g_object_unref (parser);
00951 #if DEBUG_INPUT
00952   fprintf (stderr, "input_open_json:  end\n");
00953 #endif
00954   return 0;
00955 }
00956
00962 int
00963 input_open (char *filename)
00964 {
00965   xmlDoc *doc;
00966   JsonParser *parser;
```

```
00967
00968 #if DEBUG_INPUT
00969   fprintf (stderr, "input_open:  start\n");
00970 #endif
00971
00972   // Resetting input data
00973   input_new ();
00974
00975   // Opening input file
00976 #if DEBUG_INPUT
00977   fprintf (stderr, "input_open:  opening the input file %s\n", filename);
00978   fprintf (stderr, "input_open:  trying XML format\n");
00979 #endif
00980   doc = xmlParseFile (filename);
00981   if (!doc)
00982     {
00983 #if DEBUG_INPUT
00984       fprintf (stderr, "input_open:  trying JSON format\n");
00985 #endif
00986       parser = json_parser_new ();
00987       if (!json_parser_load_from_file (parser, filename, NULL))
00988         {
00989           input_error (_("Unable to parse the input file"));
00990           goto exit_on_error;
00991         }
00992       if (!input_open_json (parser))
00993         goto exit_on_error;
00994     }
00995   else if (!input_open_xml (doc))
00996     goto exit_on_error;
00997
00998   // Getting the working directory
00999   input->directory = g_path_get_dirname (filename);
01000   input->name = g_path_get_basename (filename);
01001
01002 #if DEBUG_INPUT
01003   fprintf (stderr, "input_open:  end\n");
01004 #endif
01005   return 1;
01006
01007 exit_on_error:
01008   jbw_show_error (error_message);
01009   g_free (error_message);
01010   input_free ();
01011 #if DEBUG_INPUT
01012   fprintf (stderr, "input_open:  end\n");
01013 #endif
01014   return 0;
01015 }
```

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

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: $\max_i \lvert w_i\,x_i\rvert$.
ERROR_NORM_P	P-norm $\sqrt[p]{\sum_i \lvert w_i\,x_i\rvert^p}$.
ERROR_NORM_TAXICAB	Taxicab norm $\sum_i \lvert w_i\,x_i\rvert$.

Definition at line 49 of file input.h.

```
00050 {
00051   ERROR_NORM_EUCLIDIAN = 0,
00053   ERROR_NORM_MAXIMUM = 1,
00055   ERROR_NORM_P = 2,
00057   ERROR_NORM_TAXICAB = 3
00059 };
```

4.9.3 Function Documentation

4.9.3.1 input_free()

```
void input_free ( )
```

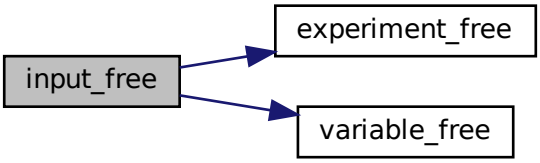
Function to free the memory of the input file data.

Definition at line 84 of file input.c.

```
00085 {
00086   unsigned int i;
00087 #if DEBUG_INPUT
00088   fprintf (stderr, "input_free:  start\n");
00089 #endif
00090   g_free (input->name);
00091   g_free (input->directory);
```

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

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 = 0;
00072   input->simulator = input->evaluator = input->directory = input->name = NULL;
00073   input->experiment = NULL;
00074   input->variable = NULL;
00075 #if DEBUG_INPUT
00076   fprintf (stderr, "input_new:  end\n");
00077 #endif
00078 }
```

4.9.3.3 input_open()

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

Function to open the input file.

Returns

1_on_success, 0_on_error.

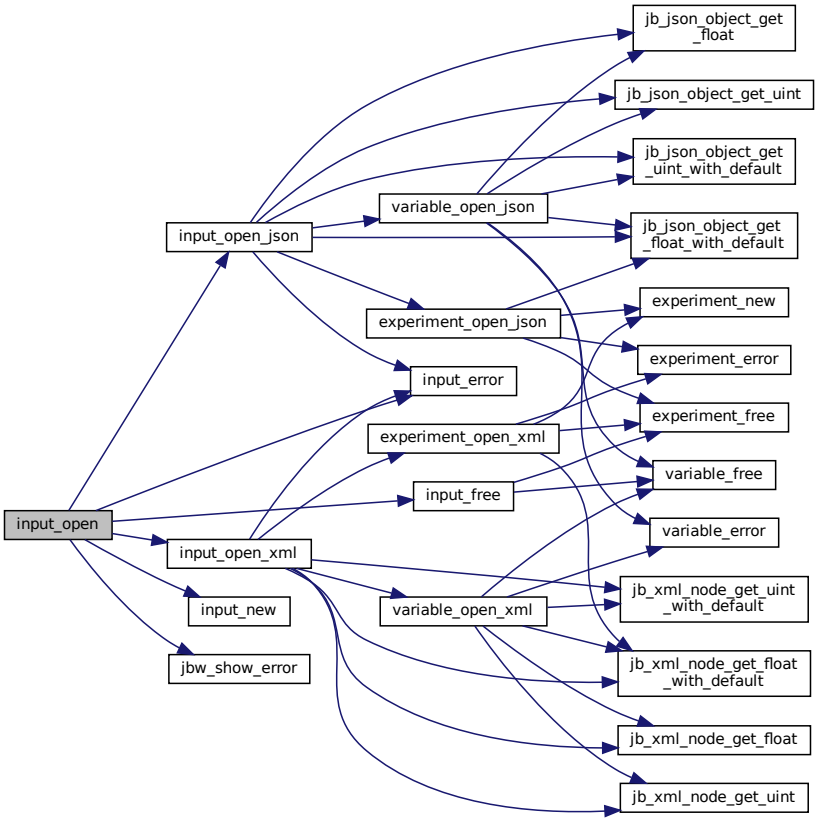
Parameters

filename	Input data file name.

Definition at line 963 of file input.c.

```
00964 {
00965   xmlDoc *doc;
00966   JsonParser *parser;
00967
00968 #if DEBUG_INPUT
00969   fprintf (stderr, "input_open:  start\n");
00970 #endif
00971
00972   // Resetting input data
00973   input_new ();
00974
00975   // Opening input file
00976 #if DEBUG_INPUT
00977   fprintf (stderr, "input_open:  opening the input file %s\n", filename);
00978   fprintf (stderr, "input_open:  trying XML format\n");
00979 #endif
00980   doc = xmlParseFile (filename);
00981   if (!doc)
00982     {
00983 #if DEBUG_INPUT
00984       fprintf (stderr, "input_open:  trying JSON format\n");
00985 #endif
00986       parser = json_parser_new ();
00987       if (!json_parser_load_from_file (parser, filename, NULL))
00988         {
00989           input_error (_("Unable to parse the input file"));
00990           goto exit_on_error;
00991         }
00992       if (!input_open_json (parser))
00993         goto exit_on_error;
00994     }
00995   else if (!input_open_xml (doc))
00996     goto exit_on_error;
00997
00998   // Getting the working directory
00999   input->directory = g_path_get_dirname (filename);
01000   input->name = g_path_get_basename (filename);
01001
01002 #if DEBUG_INPUT
01003   fprintf (stderr, "input_open:  end\n");
01004 #endif
01005   return 1;
01006
01007 exit_on_error:
01008   jbw_show_error (error_message);
01009   g_free (error_message);
01010   input_free ();
01011 #if DEBUG_INPUT
01012   fprintf (stderr, "input_open:  end\n");
01013 #endif
01014   return 0;
01015 }
```

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 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
00038 #ifndef INPUT__H
00039 #define INPUT__H 1
00040
00042 enum ClimbingMethod
00043 {
00044   CLIMBING_METHOD_COORDINATES = 0,
00045   CLIMBING_METHOD_RANDOM = 1,
00046 };
00047
00049 enum ErrorNorm
00050 {
00051   ERROR_NORM_EUCLIDIAN = 0,
00053   ERROR_NORM_MAXIMUM = 1,
00055   ERROR_NORM_P = 2,
00057   ERROR_NORM_TAXICAB = 3
00059 };
00060
00065 typedef struct
00066 {
00067   Experiment *experiment;
00068   Variable *variable;
00069   char *result;
00070   char *variables;
00071   char *simulator;
00072   char *evaluator;
00074   char *directory;
00075   char *name;
00076   double tolerance;
00077   double mutation_ratio;
00078   double reproduction_ratio;
00079   double adaptation_ratio;
00080   double relaxation;
00081   double p;
00082   double threshold;
00083   unsigned long int seed;
```

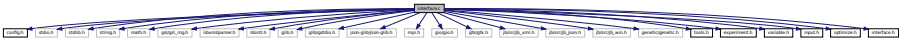
```
00085   unsigned int nvariables;
00086   unsigned int nexperiments;
00087   unsigned int nsimulations;
00088   unsigned int algorithm;
00089   unsigned int nsteps;
00091   unsigned int climbing;
00092   unsigned int nestimates;
00094   unsigned int niterations;
00095   unsigned int nbest;
00096   unsigned int norm;
00097   unsigned int type;
00098 } Input;
00099
00100 extern Input input[1];
00101 extern const char *result_name;
00102 extern const char *variables_name;
00103
00104 // Public functions
00105 void input_new ();
00106 void input_free ();
00107 int input_open (char *filename);
00108
00109 #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 ()
- 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

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_evaluator()

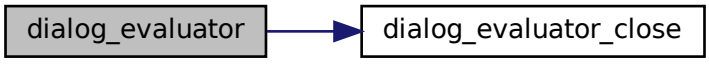
```
static void dialog_evaluator ( )  [static]
```

Function to open a dialog to save the evaluator file.

Definition at line 2251 of file interface.c.

```
02252 {
02253   GtkFileChooserDialog *dlg;
02254 #if DEBUG_INTERFACE
02255   fprintf (stderr, "dialog_evaluator:  start\n");
02256 #endif
02257   dlg = (GtkFileChooserDialog *)
02258     gtk_file_chooser_dialog_new (_("Open evaluator file"),
02259                                  window->window,
02260                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02261                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02262                                  _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02263   g_signal_connect (dlg, "response", G_CALLBACK (dialog_evaluator_close), NULL);
02264   gtk_window_present (GTK_WINDOW (dlg));
02265 #if DEBUG_INTERFACE
02266   fprintf (stderr, "dialog_evaluator:  end\n");
02267 #endif
02268 }
```

Here is the call graph for this function:



4.11.3.2 dialog_evaluator_close()

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

Function to save the close the evaluator file dialog.

Parameters

dlg	GtkFileChooserDialog dialog.
response_id	Response identifier.

Definition at line 2220 of file interface.c.

```
02223 {
02224   GFile *file1, *file2;
02225   char *buffer1, *buffer2;
02226 #if DEBUG_INTERFACE
02227   fprintf (stderr, "dialog_evaluator_close:  start\n");
02228 #endif
02229   if (response_id == GTK_RESPONSE_OK)
02230     {
02231       buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02232       file1 = g_file_new_for_path (buffer1);
02233       file2 = g_file_new_for_path (input->directory);
02234       buffer2 = g_file_get_relative_path (file2, file1);
02235       input->evaluator = g_strdup (buffer2);
02236       g_free (buffer2);
02237       g_object_unref (file2);
02238       g_object_unref (file1);
02239       g_free (buffer1);
02240     }
02241   gtk_window_destroy (GTK_WINDOW (dlg));
02242 #if DEBUG_INTERFACE
02243   fprintf (stderr, "dialog_evaluator_close:  end\n");
02244 #endif
02245 }
```

4.11.3.3 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

dlg	GtkFileChooserDialog struct.
response_id	Response identifier.
data	Function data.

Definition at line 1466 of file interface.c.

```
01470 {
01471   char *buffer;
01472   unsigned int i;
01473 #if DEBUG_INTERFACE
01474   fprintf (stderr, "window_name_experiment_close:  start\n");
01475 #endif
01476   i = (size_t) data;
01477   if (response_id == GTK_RESPONSE_OK)
01478     {
01479       buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01480       g_signal_handler_block (window->combo_experiment, window->id_experiment);
01481       gtk_combo_box_text_remove (window->combo_experiment, i);
01482       gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01483       gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01484       g_signal_handler_unblock (window->combo_experiment,
01485                                 window->id_experiment);
01486       g_free (buffer);
01487     }
01488 #if DEBUG_INTERFACE
01489   fprintf (stderr, "window_name_experiment_close:  end\n");
01490 #endif
01491 }
```

4.11.3.4 dialog_open_close()

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

Function to close the input data dialog.

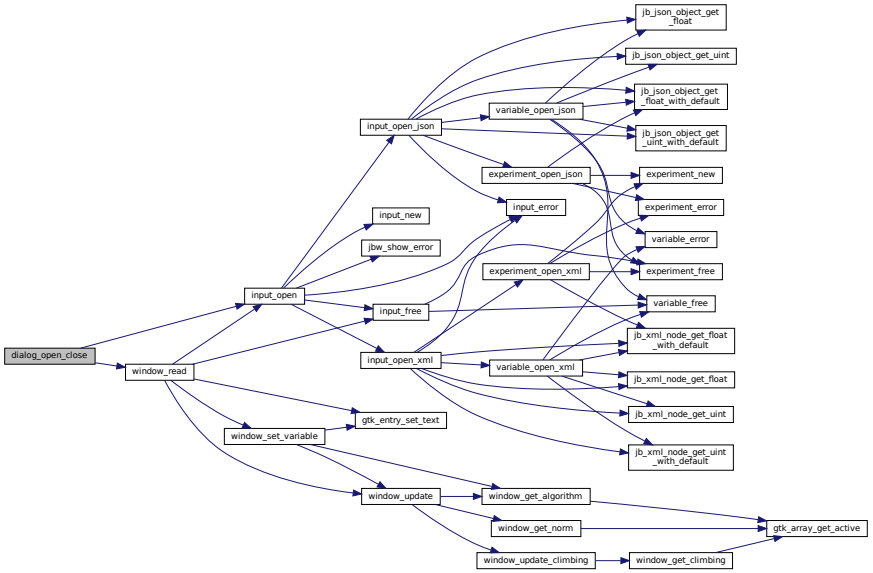
Parameters

dlg	GtkFileChooserDialog dialog.
response_id	Response identifier.

Definition at line 2060 of file interface.c.

```
02063 {
02064   char *buffer, *directory, *name;
02065
02066 #if DEBUG_INTERFACE
02067   fprintf (stderr, "dialog_open_close:  start\n");
02068 #endif
02069
02070   // Saving a backup of the current input file
02071   directory = g_strdup (input->directory);
02072   name = g_strdup (input->name);
02073
02074   // If OK saving
02075   if (response_id == GTK_RESPONSE_OK)
02076     {
02077
02078       // Traying to open the input file
02079       buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02080       if (!window_read (buffer))
02081         {
02082 #if DEBUG_INTERFACE
02083           fprintf (stderr, "dialog_open_close:  error reading input file\n");
02084 #endif
02085           g_free (buffer);
02086
02087           // Reading backup file on error
02088           buffer = g_build_filename (directory, name, NULL);
02089           input->result = input->variables = NULL;
02090           if (!input_open (buffer))
02091             {
02092
02093               // Closing on backup file reading error
02094 #if DEBUG_INTERFACE
02095               fprintf (stderr,
02096                        "dialog_open_close:  error reading backup file\n");
02097 #endif
02098             }
02099         }
02100       g_free (buffer);
02101     }
02102
02103   // Freeing and closing
02104   g_free (name);
02105   g_free (directory);
02106   gtk_window_destroy (GTK_WINDOW (dlg));
02107
02108 #if DEBUG_INTERFACE
02109   fprintf (stderr, "dialog_open_close:  end\n");
02110 #endif
02111
02112 }
```

Here is the call graph for this function:



4.11.3.5 dialog_options_close()

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

Function to close the options dialog.

Parameters

dlg	GtkDialog options dialog.
response_id	Response identifier.

Definition at line 638 of file interface.c.

```
00640 {
00641 #if DEBUG_INTERFACE
00642   fprintf (stderr, "dialog_options_close:  start\n");
00643 #endif
00644   if (response_id == GTK_RESPONSE_OK)
00645     {
00646       input->seed
00647         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00648       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00649       nthreads_climbing
00650         = gtk_spin_button_get_value_as_int (options->spin_climbing);
00651     }
00652   gtk_window_destroy (GTK_WINDOW (dlg));
00653 #if DEBUG_INTERFACE
00654   fprintf (stderr, "dialog_options_close:  end\n");
00655 #endif
00656 }
```

4.11.3.6 dialog_save_close()

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

Function to close the save dialog.

Parameters

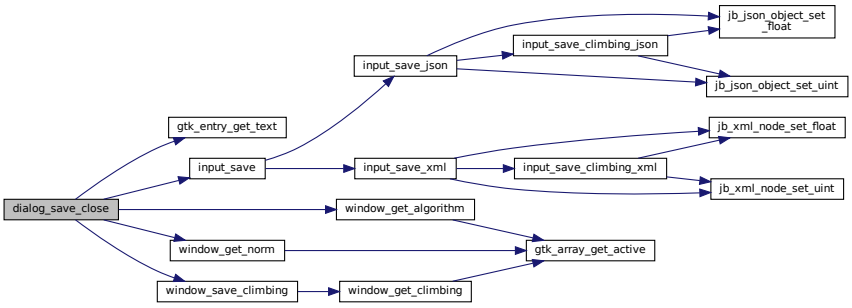
dlg	GtkFileChooserDialog dialog.
response_id	Response identifier.

Definition at line 852 of file interface.c.

```
00855 {
00856   GtkFileFilter *filter1;
00857   char *buffer;
00858 #if DEBUG_INTERFACE
00859   fprintf (stderr, "dialog_save_close:  start\n");
00860 #endif
00861   // If OK response then saving
00862   if (response_id == GTK_RESPONSE_OK)
00863     {
00864       // Setting input file type
00865       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866       buffer = (char *) gtk_file_filter_get_name (filter1);
00867       if (!strcmp (buffer, "XML"))
00868         input->type = INPUT_TYPE_XML;
00869       else
00870         input->type = INPUT_TYPE_JSON;
00871
00872       // Adding properties to the root XML node
00873       input->simulator
00874         = g_strdup (gtk_button_get_label (window->button_simulator));
00875       if (gtk_check_button_get_active (window->check_evaluator))
00876         input->evaluator
00877           = g_strdup (gtk_button_get_label (window->button_evaluator));
00878       else
00879         input->evaluator = NULL;
00880       if (input->type == INPUT_TYPE_XML)
00881         {
00882           input->result
00883             = (char *) xmlStrdup ((const xmlChar *)
00884                                   gtk_entry_get_text (window->entry_result));
00885           input->variables
00886             = (char *) xmlStrdup ((const xmlChar *)
00887                                   gtk_entry_get_text (window->entry_variables));
00888         }
00889       else
00890         {
00891           input->result = g_strdup (gtk_entry_get_text (window->entry_result));
00892           input->variables =
00893             g_strdup (gtk_entry_get_text (window->entry_variables));
00894         }
00895
00896       // Setting the algorithm
00897       switch (window_get_algorithm ())
00898         {
00899         case ALGORITHM_MONTE_CARLO:
00900           input->algorithm = ALGORITHM_MONTE_CARLO;
00901           input->nsimulations
00902             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00903           input->niterations
00904             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00905           input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00906           input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00907           window_save_climbing ();
00908           break;
00909         case ALGORITHM_SWEEP:
00910           input->algorithm = ALGORITHM_SWEEP;
00911           input->niterations
00912             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00913           input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00914           input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00915           window_save_climbing ();
00916           break;
```

```
00917         case ALGORITHM_ORTHOGONAL:
00918           input->algorithm = ALGORITHM_ORTHOGONAL;
00919           input->niterations
00920             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921           input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00922           input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00923           window_save_climbing ();
00924           break;
00925         default:
00926           input->algorithm = ALGORITHM_GENETIC;
00927           input->nsimulations
00928             = gtk_spin_button_get_value_as_int (window->spin_population);
00929           input->niterations
00930             = gtk_spin_button_get_value_as_int (window->spin_generations);
00931           input->mutation_ratio
00932             = gtk_spin_button_get_value (window->spin_mutation);
00933           input->reproduction_ratio
00934             = gtk_spin_button_get_value (window->spin_reproduction);
00935           input->adaptation_ratio
00936             = gtk_spin_button_get_value (window->spin_adaptation);
00937         }
00938       input->norm = window_get_norm ();
00939       input->p = gtk_spin_button_get_value (window->spin_p);
00940       input->threshold = gtk_spin_button_get_value (window->spin_threshold);
00941
00942       // Saving the XML file
00943       buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
00944       input_save (buffer);
00945
00946       // Closing and freeing memory
00947       g_free (buffer);
00948     }
00949   gtk_window_destroy (GTK_WINDOW (dlg));
00950 #if DEBUG_INTERFACE
00951   fprintf (stderr, "dialog_save_close:  end\n");
00952 #endif
00953 }
```

Here is the call graph for this function:



4.11.3.7 dialog_simulator()

```
static void dialog_simulator ( )  [static]
```

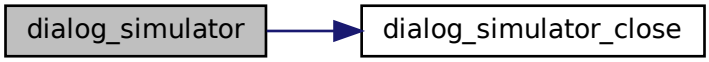
Function to open a dialog to save the simulator file.

Definition at line 2197 of file interface.c.

```
02198 {
02199   GtkFileChooserDialog *dlg;
02200 #if DEBUG_INTERFACE
02201   fprintf (stderr, "dialog_simulator:  start\n");
02202 #endif
02203   dlg = (GtkFileChooserDialog *)
02204     gtk_file_chooser_dialog_new (_("Open simulator file"),
```

```
02205                                 window->window,
02206                                 GTK_FILE_CHOOSER_ACTION_OPEN,
02207                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
02208                                 _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02209   g_signal_connect (dlg, "response", G_CALLBACK (dialog_simulator_close), NULL);
02210   gtk_window_present (GTK_WINDOW (dlg));
02211 #if DEBUG_INTERFACE
02212   fprintf (stderr, "dialog_simulator:  end\n");
02213 #endif
02214 }
```

Here is the call graph for this function:



4.11.3.8 dialog_simulator_close()

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

Function to save the close the simulator file dialog.

Parameters

dlg	GtkFileChooserDialog dialog.
response_id	Response identifier.

Definition at line 2166 of file interface.c.

```
02169 {
02170   GFile *file1, *file2;
02171   char *buffer1, *buffer2;
02172 #if DEBUG_INTERFACE
02173   fprintf (stderr, "dialog_simulator_close:  start\n");
02174 #endif
02175   if (response_id == GTK_RESPONSE_OK)
02176     {
02177       buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02178       file1 = g_file_new_for_path (buffer1);
02179       file2 = g_file_new_for_path (input->directory);
02180       buffer2 = g_file_get_relative_path (file2, file1);
02181       input->simulator = g_strdup (buffer2);
02182       g_free (buffer2);
02183       g_object_unref (file2);
02184       g_object_unref (file1);
02185       g_free (buffer1);
02186     }
02187   gtk_window_destroy (GTK_WINDOW (dlg));
02188 #if DEBUG_INTERFACE
02189   fprintf (stderr, "dialog_simulator_close:  end\n");
02190 #endif
02191 }
```

4.11.3.9 input_save()

```
static void input_save (
            char * filename )  [inline], [static]
```

Function to save the input file.

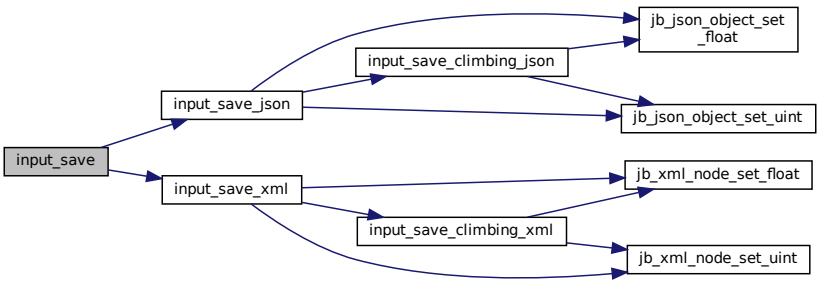
Parameters

filename	Input file name.

Definition at line 590 of file interface.c.

```
00591 {
00592   xmlDoc *doc;
00593   JsonGenerator *generator;
00594
00595 #if DEBUG_INTERFACE
00596   fprintf (stderr, "input_save:  start\n");
00597 #endif
00598
00599   // Getting the input file directory
00600   input->name = g_path_get_basename (filename);
00601   input->directory = g_path_get_dirname (filename);
00602
00603   if (input->type == INPUT_TYPE_XML)
00604     {
00605       // Opening the input file
00606       doc = xmlNewDoc ((const xmlChar *) "1.0");
00607       input_save_xml (doc);
00608
00609       // Saving the XML file
00610       xmlSaveFormatFile (filename, doc, 1);
00611
00612       // Freeing memory
00613       xmlFreeDoc (doc);
00614     }
00615   else
00616     {
00617       // Opening the input file
00618       generator = json_generator_new ();
00619       json_generator_set_pretty (generator, TRUE);
00620       input_save_json (generator);
00621
00622       // Saving the JSON file
00623       json_generator_to_file (generator, filename, NULL);
00624
00625       // Freeing memory
00626       g_object_unref (generator);
00627     }
00628
00629 #if DEBUG_INTERFACE
00630   fprintf (stderr, "input_save:  end\n");
00631 #endif
00632 }
```

Here is the call graph for this function:



4.11.3.10 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.

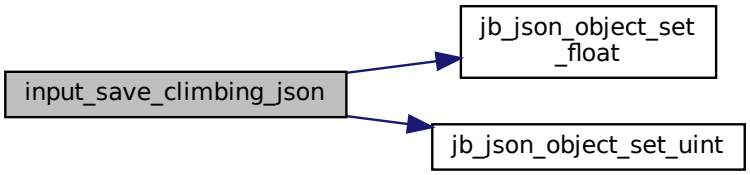
Parameters

node	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:



4.11.3.11 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.

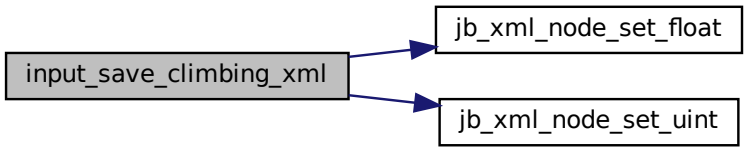
Parameters

node	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:



4.11.3.12 input_save_json()

```
static void input_save_json (
             JsonGenerator * generator )  [inline], [static]
```

Function to save the input file in JSON format.

Parameters

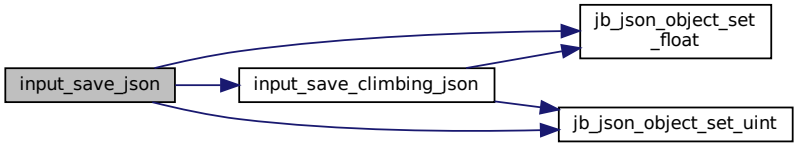
generator	JsonGenerator struct.

Definition at line 418 of file interface.c.

```
00419 {
00420   unsigned int i, j;
00421   char *buffer;
00422   JsonNode *node, *child;
00423   JsonObject *object;
00424   JsonArray *array;
00425   GFile *file, *file2;
00426
00427 #if DEBUG_INTERFACE
00428   fprintf (stderr, "input_save_json:  start\n");
00429 #endif
00430
00431   // Setting root JSON node
00432   object = json_object_new ();
00433   node = json_node_new (JSON_NODE_OBJECT);
00434   json_node_set_object (node, object);
00435   json_generator_set_root (generator, node);
00436
00437   // Adding properties to the root JSON node
00438   if (strcmp (input->result, result_name))
00439     json_object_set_string_member (object, LABEL_RESULT_FILE, input->result);
00440   if (strcmp (input->variables, variables_name))
00441     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00442                                    input->variables);
00443   file = g_file_new_for_path (input->directory);
00444   file2 = g_file_new_for_path (input->simulator);
00445   buffer = g_file_get_relative_path (file, file2);
00446   g_object_unref (file2);
00447   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00448   g_free (buffer);
00449   if (input->evaluator)
00450     {
00451       file2 = g_file_new_for_path (input->evaluator);
00452       buffer = g_file_get_relative_path (file, file2);
00453       g_object_unref (file2);
00454       if (strlen (buffer))
00455         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00456       g_free (buffer);
00457     }
00458   if (input->seed != DEFAULT_RANDOM_SEED)
00459     jb_json_object_set_uint (object, LABEL_SEED, input->seed);
00460
00461   // Setting the algorithm
00462   buffer = (char *) g_slice_alloc (64);
00463   switch (input->algorithm)
00464     {
00465     case ALGORITHM_MONTE_CARLO:
00466       json_object_set_string_member (object, LABEL_ALGORITHM,
00467                                      LABEL_MONTE_CARLO);
00468       snprintf (buffer, 64, "%u", input->nsimulations);
00469       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00470       snprintf (buffer, 64, "%u", input->niterations);
00471       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00472       snprintf (buffer, 64, "%.3lg", input->tolerance);
00473       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00474       snprintf (buffer, 64, "%u", input->nbest);
00475       json_object_set_string_member (object, LABEL_NBEST, buffer);
00476       input_save_climbing_json (node);
00477       break;
00478     case ALGORITHM_SWEEP:
00479       json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_SWEEP);
00480       snprintf (buffer, 64, "%u", input->niterations);
00481       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00482       snprintf (buffer, 64, "%.3lg", input->tolerance);
00483       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00484       snprintf (buffer, 64, "%u", input->nbest);
00485       json_object_set_string_member (object, LABEL_NBEST, buffer);
00486       input_save_climbing_json (node);
00487       break;
00488     case ALGORITHM_ORTHOGONAL:
00489       json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_ORTHOGONAL);
00490       snprintf (buffer, 64, "%u", input->niterations);
00491       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00492       snprintf (buffer, 64, "%.3lg", input->tolerance);
00493       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00494       snprintf (buffer, 64, "%u", input->nbest);
00495       json_object_set_string_member (object, LABEL_NBEST, buffer);
00496       input_save_climbing_json (node);
00497       break;
00498     default:
00499       json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_GENETIC);
00500       snprintf (buffer, 64, "%u", input->nsimulations);
00501       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00502       snprintf (buffer, 64, "%u", input->niterations);
00503       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00504       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00505       json_object_set_string_member (object, LABEL_MUTATION, buffer);
```

```
00506       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00507       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00508       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00509       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00510       break;
00511     }
00512   g_slice_free1 (64, buffer);
00513   if (input->threshold != 0.)
00514     jb_json_object_set_float (object, LABEL_THRESHOLD, input->threshold);
00515
00516   // Setting the experimental data
00517   array = json_array_new ();
00518   for (i = 0; i < input->nexperiments; ++i)
00519     {
00520       child = json_node_new (JSON_NODE_OBJECT);
00521       object = json_node_get_object (child);
00522       json_object_set_string_member (object, LABEL_NAME,
00523                                      input->experiment[i].name);
00524       if (input->experiment[i].weight != 1.)
00525         jb_json_object_set_float (object, LABEL_WEIGHT,
00526                                   input->experiment[i].weight);
00527       for (j = 0; j < input->experiment->ninputs; ++j)
00528         json_object_set_string_member (object, stencil[j],
00529                                        input->experiment[i].stencil[j]);
00530       json_array_add_element (array, child);
00531     }
00532   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00533
00534   // Setting the variables data
00535   array = json_array_new ();
00536   for (i = 0; i < input->nvariables; ++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->variable[i].name);
00542       jb_json_object_set_float (object, LABEL_MINIMUM,
00543                                 input->variable[i].rangemin);
00544       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00545         jb_json_object_set_float (object, LABEL_ABSOLUTE_MINIMUM,
00546                                   input->variable[i].rangeminabs);
00547       jb_json_object_set_float (object, LABEL_MAXIMUM,
00548                                 input->variable[i].rangemax);
00549       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00550         jb_json_object_set_float (object, LABEL_ABSOLUTE_MAXIMUM,
00551                                   input->variable[i].rangemaxabs);
00552       if (input->variable[i].precision != DEFAULT_PRECISION)
00553         jb_json_object_set_uint (object, LABEL_PRECISION,
00554                                  input->variable[i].precision);
00555       if (input->algorithm == ALGORITHM_SWEEP
00556           || input->algorithm == ALGORITHM_ORTHOGONAL)
00557         jb_json_object_set_uint (object, LABEL_NSWEEPS,
00558                                  input->variable[i].nsweeps);
00559       else if (input->algorithm == ALGORITHM_GENETIC)
00560         jb_json_object_set_uint (object, LABEL_NBITS, input->variable[i].nbits);
00561       if (input->nsteps)
00562         jb_json_object_set_float (object, LABEL_STEP, input->variable[i].step);
00563       json_array_add_element (array, child);
00564     }
00565   json_object_set_array_member (object, LABEL_VARIABLES, array);
00566
00567   // Saving the error norm
00568   switch (input->norm)
00569     {
00570     case ERROR_NORM_MAXIMUM:
00571       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00572       break;
00573     case ERROR_NORM_P:
00574       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00575       jb_json_object_set_float (object, LABEL_P, input->p);
00576       break;
00577     case ERROR_NORM_TAXICAB:
00578       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00579     }
00580
00581 #if DEBUG_INTERFACE
00582   fprintf (stderr, "input_save_json:  end\n");
00583 #endif
00584 }
```

Here is the call graph for this function:



4.11.3.13 input_save_xml()

```
static void input_save_xml (
             xmlDoc * doc )  [inline], [static]
```

Function to save the input file in XML format.

Parameters

doc	xmlDoc struct.

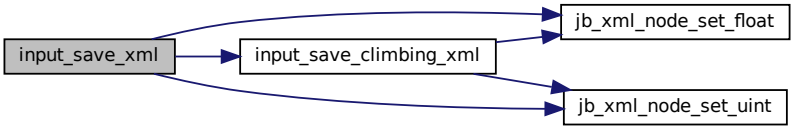
Definition at line 238 of file interface.c.

```
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->seed != DEFAULT_RANDOM_SEED)
00279     jb_xml_node_set_uint (node, (const xmlChar *) LABEL_SEED, input->seed);
```

```
00280
00281   // Setting the algorithm
00282   buffer = (char *) g_slice_alloc (64);
00283   switch (input->algorithm)
00284     {
00285     case ALGORITHM_MONTE_CARLO:
00286       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00287                   (const xmlChar *) LABEL_MONTE_CARLO);
00288       snprintf (buffer, 64, "%u", input->nsimulations);
00289       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00290                   (xmlChar *) buffer);
00291       snprintf (buffer, 64, "%u", input->niterations);
00292       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00293                   (xmlChar *) buffer);
00294       snprintf (buffer, 64, "%.3lg", input->tolerance);
00295       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00296       snprintf (buffer, 64, "%u", input->nbest);
00297       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00298       input_save_climbing_xml (node);
00299       break;
00300     case ALGORITHM_SWEEP:
00301       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00302                   (const xmlChar *) LABEL_SWEEP);
00303       snprintf (buffer, 64, "%u", input->niterations);
00304       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00305                   (xmlChar *) buffer);
00306       snprintf (buffer, 64, "%.3lg", input->tolerance);
00307       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00308       snprintf (buffer, 64, "%u", input->nbest);
00309       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00310       input_save_climbing_xml (node);
00311       break;
00312     case ALGORITHM_ORTHOGONAL:
00313       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00314                   (const xmlChar *) LABEL_ORTHOGONAL);
00315       snprintf (buffer, 64, "%u", input->niterations);
00316       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00317                   (xmlChar *) buffer);
00318       snprintf (buffer, 64, "%.3lg", input->tolerance);
00319       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00320       snprintf (buffer, 64, "%u", input->nbest);
00321       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00322       input_save_climbing_xml (node);
00323       break;
00324     default:
00325       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00326                   (const xmlChar *) LABEL_GENETIC);
00327       snprintf (buffer, 64, "%u", input->nsimulations);
00328       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00329                   (xmlChar *) buffer);
00330       snprintf (buffer, 64, "%u", input->niterations);
00331       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00332                   (xmlChar *) buffer);
00333       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00334       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00335       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00336       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00337                   (xmlChar *) buffer);
00338       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00339       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00340       break;
00341     }
00342   g_slice_free1 (64, buffer);
00343   if (input->threshold != 0.)
00344     jb_xml_node_set_float (node, (const xmlChar *) LABEL_THRESHOLD,
00345                            input->threshold);
00346
00347   // Setting the experimental data
00348   for (i = 0; i < input->nexperiments; ++i)
00349     {
00350       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00351       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00352                   (xmlChar *) input->experiment[i].name);
00353       if (input->experiment[i].weight != 1.)
00354         jb_xml_node_set_float (child, (const xmlChar *) LABEL_WEIGHT,
00355                                input->experiment[i].weight);
00356       for (j = 0; j < input->experiment->ninputs; ++j)
00357         xmlSetProp (child, (const xmlChar *) stencil[j],
00358                     (xmlChar *) input->experiment[i].stencil[j]);
00359     }
00360
00361   // Setting the variables data
00362   for (i = 0; i < input->nvariables; ++i)
00363     {
00364       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00365       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00366                   (xmlChar *) input->variable[i].name);
```

```
00367       jb_xml_node_set_float (child, (const xmlChar *) LABEL_MINIMUM,
00368                              input->variable[i].rangemin);
00369       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00370         jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM,
00371                                input->variable[i].rangeminabs);
00372       jb_xml_node_set_float (child, (const xmlChar *) LABEL_MAXIMUM,
00373                              input->variable[i].rangemax);
00374       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00375         jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM,
00376                                input->variable[i].rangemaxabs);
00377       if (input->variable[i].precision != DEFAULT_PRECISION)
00378         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_PRECISION,
00379                               input->variable[i].precision);
00380       if (input->algorithm == ALGORITHM_SWEEP
00381           || input->algorithm == ALGORITHM_ORTHOGONAL)
00382         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NSWEEPS,
00383                               input->variable[i].nsweeps);
00384       else if (input->algorithm == ALGORITHM_GENETIC)
00385         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00386                               input->variable[i].nbits);
00387       if (input->nsteps)
00388         jb_xml_node_set_float (child, (const xmlChar *) LABEL_STEP,
00389                                input->variable[i].step);
00390     }
00391
00392   // Saving the error norm
00393   switch (input->norm)
00394     {
00395     case ERROR_NORM_MAXIMUM:
00396       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397                   (const xmlChar *) LABEL_MAXIMUM);
00398       break;
00399     case ERROR_NORM_P:
00400       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                   (const xmlChar *) LABEL_P);
00402       jb_xml_node_set_float (node, (const xmlChar *) LABEL_P, input->p);
00403       break;
00404     case ERROR_NORM_TAXICAB:
00405       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00406                   (const xmlChar *) LABEL_TAXICAB);
00407     }
00408
00409 #if DEBUG_INTERFACE
00410   fprintf (stderr, "input_save:  end\n");
00411 #endif
00412 }
```

Here is the call graph for this function:



4.11.3.14 options_new()

```
static void options_new ( )  [static]
```

Function to open the options dialog.

Definition at line 662 of file interface.c.

```
00663 {
00664 #if DEBUG_INTERFACE
00665   fprintf (stderr, "options_new:  start\n");
00666 #endif
```

```
00667   options->label_seed = (GtkLabel *)
00668     gtk_label_new (_("Pseudo-random numbers generator seed"));
00669   options->spin_seed = (GtkSpinButton *)
00670     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00671   gtk_widget_set_tooltip_text
00672     (GTK_WIDGET (options->spin_seed),
00673      _("Seed to init the pseudo-random numbers generator"));
00674   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
00675   options->label_threads = (GtkLabel *)
00676     gtk_label_new (_("Threads number for the stochastic algorithm"));
00677   options->spin_threads
00678     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00679   gtk_widget_set_tooltip_text
00680     (GTK_WIDGET (options->spin_threads),
00681      _("Number of threads to perform the calibration/optimization for "
00682        "the stochastic algorithm"));
00683   gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00684   options->label_climbing = (GtkLabel *)
00685     gtk_label_new (_("Threads number for the hill climbing method"));
00686   options->spin_climbing =
00687     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00688   gtk_widget_set_tooltip_text
00689     (GTK_WIDGET (options->spin_climbing),
00690      _("Number of threads to perform the calibration/optimization for the "
00691        "hill climbing method"));
00692   gtk_spin_button_set_value (options->spin_climbing,
00693                              (gdouble) nthreads_climbing);
00694   options->grid = (GtkGrid *) gtk_grid_new ();
00695   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00696   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00697   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00698                    0, 1, 1, 1);
00699   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00700                    1, 1, 1, 1);
00701   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
00702                    1);
00703   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
00704                    1);
00705 #if !GTK4
00706   gtk_widget_show_all (GTK_WIDGET (options->grid));
00707 #else
00708   gtk_widget_show (GTK_WIDGET (options->grid));
00709 #endif
00710   options->dialog = (GtkDialog *)
00711     gtk_dialog_new_with_buttons (_("Options"),
00712                                  window->window,
00713                                  GTK_DIALOG_MODAL,
00714                                  _("_OK"), GTK_RESPONSE_OK,
00715                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00716   gtk_box_append (GTK_BOX (gtk_dialog_get_content_area (options->dialog)),
00717                   GTK_WIDGET (options->grid));
00718   g_signal_connect (options->dialog, "response",
00719                     G_CALLBACK (dialog_options_close), NULL);
00720   gtk_window_present (GTK_WINDOW (options->dialog));
00721 #if DEBUG_INTERFACE
00722   fprintf (stderr, "options_new:  end\n");
00723 #endif
00724 }
```

4.11.3.15 running_new()

```
static void running_new ( )  [inline], [static]
```

Function to open the running dialog.

Definition at line 730 of file interface.c.

```
00731 {
00732 #if DEBUG_INTERFACE
00733   fprintf (stderr, "running_new:  start\n");
00734 #endif
00735   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00736   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00737   running->grid = (GtkGrid *) gtk_grid_new ();
00738   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00739   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00740   running->dialog = (GtkDialog *)
00741     gtk_dialog_new_with_buttons (_("Calculating"),
00742                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
```

```
00743   gtk_window_set_child (GTK_WINDOW
00744                         (gtk_dialog_get_content_area (running->dialog)),
00745                         GTK_WIDGET (running->grid));
00746   gtk_spinner_start (running->spinner);
00747 #if !GTK4
00748   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00749 #else
00750   gtk_widget_show (GTK_WIDGET (running->dialog));
00751 #endif
00752 #if DEBUG_INTERFACE
00753   fprintf (stderr, "running_new:  end\n");
00754 #endif
00755 }
```

4.11.3.16 window_about()

```
static void window_about ( )  [static]
```

Function to show an about dialog.

Definition at line 1102 of file interface.c.

```
01103 {
01104   static const gchar *authors[] = {
01105     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01106     "Borja Latorre Garcés <borja.latorre@csic.es>",
01107     NULL
01108   };
01109 #if DEBUG_INTERFACE
01110   fprintf (stderr, "window_about:  start\n");
01111 #endif
01112   gtk_show_about_dialog
01113     (window->window,
01114      "program_name", "MPCOTool",
01115      "comments",
01116      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01117        "A software to perform calibrations or optimizations of empirical "
01118        "parameters"),
01119      "authors", authors,
01120      "translator-credits",
01121      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01122      "(english, french and spanish)\n"
01123      "Uğur Çayoğlu (german)",
01124      "version", "4.4.1",
01125      "copyright", "Copyright 2012-2023 Javier Burguete Tolosa",
01126      "logo", window->logo,
01127      "website", "https://github.com/jburguete/mpcotool",
01128      "license-type", GTK_LICENSE_BSD, NULL);
01129 #if DEBUG_INTERFACE
01130   fprintf (stderr, "window_about:  end\n");
01131 #endif
01132 }
```

4.11.3.17 window_add_experiment()

```
static void window_add_experiment ( )  [static]
```

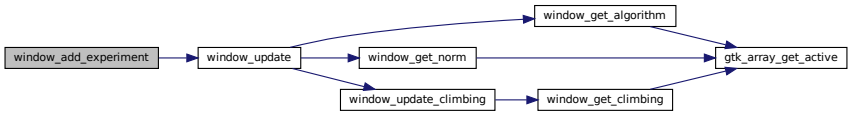
Function to add an experiment in the main window.

Definition at line 1417 of file interface.c.

```
01418 {
01419   unsigned int i, j;
01420 #if DEBUG_INTERFACE
01421   fprintf (stderr, "window_add_experiment:  start\n");
01422 #endif
01423   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01424   g_signal_handler_block (window->combo_experiment, window->id_experiment);
01425   gtk_combo_box_text_insert_text
01426     (window->combo_experiment, i, input->experiment[i].name);
```

```
01427   g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
01428   input->experiment = (Experiment *) g_realloc
01429     (input->experiment, (input->nexperiments + 1) * sizeof (Experiment));
01430   for (j = input->nexperiments - 1; j > i; --j)
01431     memcpy (input->experiment + j + 1, input->experiment + j,
01432             sizeof (Experiment));
01433   input->experiment[j + 1].weight = input->experiment[j].weight;
01434   input->experiment[j + 1].ninputs = input->experiment[j].ninputs;
01435   if (input->type == INPUT_TYPE_XML)
01436     {
01437       input->experiment[j + 1].name
01438         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
01439       for (j = 0; j < input->experiment->ninputs; ++j)
01440         input->experiment[i + 1].stencil[j]
01441           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].stencil[j]);
01442     }
01443   else
01444     {
01445       input->experiment[j + 1].name = g_strdup (input->experiment[j].name);
01446       for (j = 0; j < input->experiment->ninputs; ++j)
01447         input->experiment[i + 1].stencil[j]
01448           = g_strdup (input->experiment[i].stencil[j]);
01449     }
01450   ++input->nexperiments;
01451   for (j = 0; j < input->experiment->ninputs; ++j)
01452     g_signal_handler_block (window->button_template[j], window->id_input[j]);
01453   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01454   for (j = 0; j < input->experiment->ninputs; ++j)
01455     g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
01456   window_update ();
01457 #if DEBUG_INTERFACE
01458   fprintf (stderr, "window_add_experiment:  end\n");
01459 #endif
01460 }
```

Here is the call graph for this function:



4.11.3.18 window_add_variable()

```
static void window_add_variable ( )  [static]
```

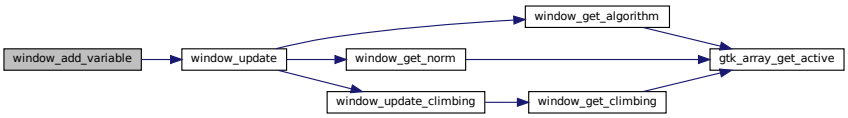
Function to add a variable in the main window.

Definition at line 1752 of file interface.c.

```
01753 {
01754   unsigned int i, j;
01755 #if DEBUG_INTERFACE
01756   fprintf (stderr, "window_add_variable:  start\n");
01757 #endif
01758   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01759   g_signal_handler_block (window->combo_variable, window->id_variable);
01760   gtk_combo_box_text_insert_text (window->combo_variable, i,
01761                                   input->variable[i].name);
01762   g_signal_handler_unblock (window->combo_variable, window->id_variable);
01763   input->variable = (Variable *) g_realloc
01764     (input->variable, (input->nvariables + 1) * sizeof (Variable));
01765   for (j = input->nvariables - 1; j > i; --j)
01766     memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01767   memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01768   if (input->type == INPUT_TYPE_XML)
01769     input->variable[j + 1].name
01770       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01771   else
```

```
01772     input->variable[j + 1].name = g_strdup (input->variable[j].name);
01773   ++input->nvariables;
01774   g_signal_handler_block (window->entry_variable, window->id_variable_label);
01775   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01776   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01777   window_update ();
01778 #if DEBUG_INTERFACE
01779   fprintf (stderr, "window_add_variable:  end\n");
01780 #endif
01781 }
```

Here is the call graph for this function:



4.11.3.19 window_get_algorithm()

```
static unsigned int window_get_algorithm ( )  [static]
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 763 of file interface.c.

```
00764 {
00765   unsigned int i;
00766 #if DEBUG_INTERFACE
00767   fprintf (stderr, "window_get_algorithm:  start\n");
00768 #endif
00769   i = gtk_array_get_active (window->button_algorithm, NALGORITHMS);
00770 #if DEBUG_INTERFACE
00771   fprintf (stderr, "window_get_algorithm:  %u\n", i);
00772   fprintf (stderr, "window_get_algorithm:  end\n");
00773 #endif
00774   return i;
00775 }
```

Here is the call graph for this function:

4.11.3.20 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 783 of file interface.c.

```
00784 {
00785   unsigned int i;
00786 #if DEBUG_INTERFACE
00787   fprintf (stderr, "window_get_climbing:  start\n");
00788 #endif
00789   i = gtk_array_get_active (window->button_climbing, NCLIMBINGS);
00790 #if DEBUG_INTERFACE
00791   fprintf (stderr, "window_get_climbing:  %u\n", i);
00792   fprintf (stderr, "window_get_climbing:  end\n");
00793 #endif
00794   return i;
00795 }
```

Here is the call graph for this function:

4.11.3.21 window_get_norm()

```
static unsigned int window_get_norm ( )  [static]
```

Function to get the norm method number.

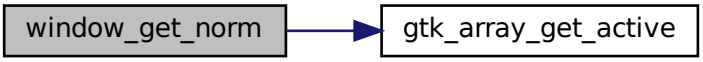
Returns

Norm method number.

Definition at line 803 of file interface.c.

```
00804 {
00805   unsigned int i;
00806 #if DEBUG_INTERFACE
00807   fprintf (stderr, "window_get_norm:  start\n");
00808 #endif
00809   i = gtk_array_get_active (window->button_norm, NNORMS);
00810 #if DEBUG_INTERFACE
00811   fprintf (stderr, "window_get_norm:  %u\n", i);
00812   fprintf (stderr, "window_get_norm:  end\n");
00813 #endif
00814   return i;
00815 }
```

Here is the call graph for this function:



4.11.3.22 window_help()

```
static void window_help ( )  [static]
```

Function to show a help dialog.

Definition at line 1070 of file interface.c.

```
01071 {
01072   char *buffer, *buffer2;
01073 #if DEBUG_INTERFACE
01074   fprintf (stderr, "window_help:  start\n");
01075 #endif
01076   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01077                               _("user-manual.pdf"), NULL);
01078   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01079   g_free (buffer2);
01080 #if !GTK4
01081 #if GTK_MINOR_VERSION >= 22
01082   gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01083 #else
01084   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01085 #endif
01086 #else
01087   gtk_show_uri (window->window, buffer, GDK_CURRENT_TIME);
01088 #endif
01089 #if DEBUG_INTERFACE
01090   fprintf (stderr, "window_help:  uri=%s\n", buffer);
01091 #endif
01092   g_free (buffer);
01093 #if DEBUG_INTERFACE
01094   fprintf (stderr, "window_help:  end\n");
01095 #endif
01096 }
```

4.11.3.23 window_inputs_experiment()

```
static void window_inputs_experiment ( )  [static]
```

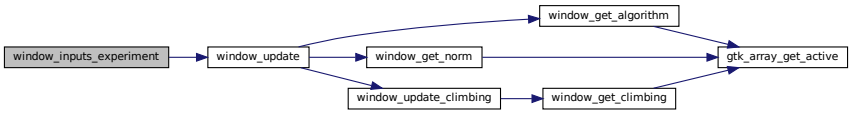
Function to update the experiment input templates number in the main window.

Definition at line 1550 of file interface.c.

```
01551 {
01552   unsigned int j;
01553 #if DEBUG_INTERFACE
01554   fprintf (stderr, "window_inputs_experiment:  start\n");
01555 #endif
01556   j = input->experiment->ninputs - 1;
01557   if (j && !gtk_check_button_get_active (window->check_template[j]))
01558     --input->experiment->ninputs;
```

```
01559   if (input->experiment->ninputs < MAX_NINPUTS
01560       && gtk_check_button_get_active (window->check_template[j]))
01561     ++input->experiment->ninputs;
01562   window_update ();
01563 #if DEBUG_INTERFACE
01564   fprintf (stderr, "window_inputs_experiment:  end\n");
01565 #endif
01566 }
```

Here is the call graph for this function:



4.11.3.24 window_label_variable()

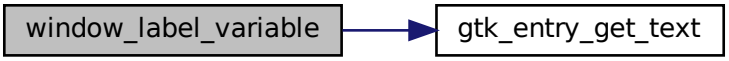
```
static void window_label_variable ( )  [static]
```

Function to set the variable label in the main window.

Definition at line 1787 of file interface.c.

```
01788 {
01789   unsigned int i;
01790   const char *buffer;
01791 #if DEBUG_INTERFACE
01792   fprintf (stderr, "window_label_variable:  start\n");
01793 #endif
01794   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01795   buffer = gtk_entry_get_text (window->entry_variable);
01796   g_signal_handler_block (window->combo_variable, window->id_variable);
01797   gtk_combo_box_text_remove (window->combo_variable, i);
01798   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01799   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01800   g_signal_handler_unblock (window->combo_variable, window->id_variable);
01801 #if DEBUG_INTERFACE
01802   fprintf (stderr, "window_label_variable:  end\n");
01803 #endif
01804 }
```

Here is the call graph for this function:



4.11.3.25 window_name_experiment()

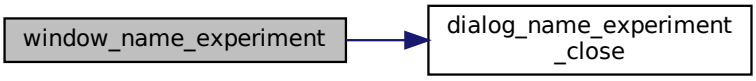
```
static void window_name_experiment ( )  [static]
```

Function to set the experiment name in the main window.

Definition at line 1497 of file interface.c.

```
01498 {
01499   GtkFileChooserDialog *dlg;
01500   GMainLoop *loop;
01501   const char *buffer;
01502   unsigned int i;
01503 #if DEBUG_INTERFACE
01504   fprintf (stderr, "window_name_experiment:  start\n");
01505 #endif
01506   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01507   buffer = gtk_button_get_label (window->button_experiment);
01508   dlg = (GtkFileChooserDialog *)
01509     gtk_file_chooser_dialog_new (_("Open experiment file"),
01510                                  window->window,
01511                                  GTK_FILE_CHOOSER_ACTION_OPEN,
01512                                  _("_Cancel"),
01513                                  GTK_RESPONSE_CANCEL,
01514                                  _("_Open"), GTK_RESPONSE_OK, NULL);
01515   gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01516   g_signal_connect (dlg, "response", G_CALLBACK (dialog_name_experiment_close),
01517                     (void *) (size_t) i);
01518   gtk_window_present (GTK_WINDOW (dlg));
01519   loop = g_main_loop_new (NULL, 0);
01520   g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01521                             loop);
01522   g_main_loop_run (loop);
01523   g_main_loop_unref (loop);
01524 #if DEBUG_INTERFACE
01525   fprintf (stderr, "window_name_experiment:  end\n");
01526 #endif
01527 }
```

Here is the call graph for this function:



4.11.3.26 window_new()

```
void window_new (
             GtkApplication * application )
```

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2274 of file interface.c.

```
02275 {
02276   unsigned int i;
02277   char *buffer, *buffer2, buffer3[64];
02278   const char *label_algorithm[NALGORITHMS] = {
02279     "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02280   };
02281   const char *tip_algorithm[NALGORITHMS] = {
02282     _("Monte-Carlo brute force algorithm"),
02283     _("Sweep brute force algorithm"),
02284     _("Genetic algorithm"),
02285     _("Orthogonal sampling brute force algorithm"),
02286   };
02287   const char *label_climbing[NCLIMBINGS] = {
02288     _("_Coordinates climbing"), _("_Random climbing")
02289   };
02290   const char *tip_climbing[NCLIMBINGS] = {
02291     _("Coordinates climbing estimate method"),
02292     _("Random climbing estimate method")
02293   };
02294   const char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02295   const char *tip_norm[NNORMS] = {
02296     _("Euclidean error norm (L2)"),
02297     _("Maximum error norm (L)"),
02298     _("P error norm (Lp)"),
02299     _("Taxicab error norm (L1)")
02300   };
02301 #if !GTK4
02302   const char *close = "delete-event";
02303 #else
02304   const char *close = "close-request";
02305 #endif
02306
02307 #if DEBUG_INTERFACE
02308   fprintf (stderr, "window_new:  start\n");
02309 #endif
02310
02311   // Creating the window
02312   window->window = main_window
02313     = (GtkWindow *) gtk_application_window_new (application);
02314
02315   // Finish when closing the window
02316   g_signal_connect_swapped (window->window, close,
02317                             G_CALLBACK (g_application_quit),
02318                             G_APPLICATION (application));
02319
02320   // Setting the window title
02321   gtk_window_set_title (window->window, "MPCOTool");
02322
02323   // Creating the open button
02324   window->button_open = (GtkButton *)
02325 #if !GTK4
02326     gtk_button_new_from_icon_name ("document-open", GTK_ICON_SIZE_BUTTON);
02327 #else
02328     gtk_button_new_from_icon_name ("document-open");
02329 #endif
02330   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02331
02332   // Creating the save button
02333   window->button_save = (GtkButton *)
02334 #if !GTK4
02335     gtk_button_new_from_icon_name ("document-save", GTK_ICON_SIZE_BUTTON);
02336 #else
02337     gtk_button_new_from_icon_name ("document-save");
02338 #endif
02339   g_signal_connect (window->button_save, "clicked", (GCallback) window_save,
02340                     NULL);
02341
02342   // Creating the run button
02343   window->button_run = (GtkButton *)
02344 #if !GTK4
02345     gtk_button_new_from_icon_name ("system-run", GTK_ICON_SIZE_BUTTON);
02346 #else
02347     gtk_button_new_from_icon_name ("system-run");
02348 #endif
02349   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02350
02351   // Creating the options button
02352   window->button_options = (GtkButton *)
02353 #if !GTK4
02354     gtk_button_new_from_icon_name ("preferences-system", GTK_ICON_SIZE_BUTTON);
02355 #else
02356     gtk_button_new_from_icon_name ("preferences-system");
02357 #endif
02358   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02359
02360   // Creating the help button
02361   window->button_help = (GtkButton *)
```

```
02362 #if !GTK4
02363     gtk_button_new_from_icon_name ("help-browser", GTK_ICON_SIZE_BUTTON);
02364 #else
02365     gtk_button_new_from_icon_name ("help-browser");
02366 #endif
02367   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02368
02369   // Creating the about button
02370   window->button_about = (GtkButton *)
02371 #if !GTK4
02372     gtk_button_new_from_icon_name ("help-about", GTK_ICON_SIZE_BUTTON);
02373 #else
02374     gtk_button_new_from_icon_name ("help-about");
02375 #endif
02376   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02377
02378   // Creating the exit button
02379   window->button_exit = (GtkButton *)
02380 #if !GTK4
02381     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
02382 #else
02383     gtk_button_new_from_icon_name ("application-exit");
02384 #endif
02385   g_signal_connect_swapped (window->button_exit, "clicked",
02386                             G_CALLBACK (g_application_quit),
02387                             G_APPLICATION (application));
02388
02389   // Creating the buttons bar
02390   window->box_buttons = (GtkBox *) gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
02391   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_open));
02392   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_save));
02393   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_run));
02394   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_options));
02395   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_help));
02396   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_about));
02397   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_exit));
02398
02399   // Creating the simulator program label and entry
02400   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02401   window->button_simulator = (GtkButton *)
02402     gtk_button_new_with_mnemonic (_("Simulator program"));
02403   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02404                                _("Simulator program executable file"));
02405   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02406   g_signal_connect (window->button_simulator, "clicked",
02407                     G_CALLBACK (dialog_simulator), NULL);
02408
02409   // Creating the evaluator program label and entry
02410   window->check_evaluator = (GtkCheckButton *)
02411     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02412   g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
02413   window->button_evaluator = (GtkButton *)
02414     gtk_button_new_with_mnemonic (_("Evaluator program"));
02415   gtk_widget_set_tooltip_text
02416     (GTK_WIDGET (window->button_evaluator),
02417      _("Optional evaluator program executable file"));
02418   g_signal_connect (window->button_evaluator, "clicked",
02419                     G_CALLBACK (dialog_evaluator), NULL);
02420
02421   // Creating the results files labels and entries
02422   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02423   window->entry_result = (GtkEntry *) gtk_entry_new ();
02424   gtk_widget_set_tooltip_text
02425     (GTK_WIDGET (window->entry_result), _("Best results file"));
02426   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02427   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02428   gtk_widget_set_tooltip_text
02429     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02430
02431   // Creating the files grid and attaching widgets
02432   window->grid_files = (GtkGrid *) gtk_grid_new ();
02433   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
02434                    0, 0, 1, 1);
02435   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
02436                    1, 0, 1, 1);
02437   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
02438                    0, 1, 1, 1);
02439   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
02440                    1, 1, 1, 1);
02441   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
02442                    0, 2, 1, 1);
02443   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
02444                    1, 2, 1, 1);
02445   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
02446                    0, 3, 1, 1);
02447   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
02448                    1, 3, 1, 1);
```

```
02449
02450   // Creating the algorithm properties
02451   window->label_simulations = (GtkLabel *) gtk_label_new
02452     (_("Simulations number"));
02453   window->spin_simulations
02454     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02455   gtk_widget_set_tooltip_text
02456     (GTK_WIDGET (window->spin_simulations),
02457      _("Number of simulations to perform for each iteration"));
02458   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02459   window->label_iterations = (GtkLabel *)
02460     gtk_label_new (_("Iterations number"));
02461   window->spin_iterations
02462     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02463   gtk_widget_set_tooltip_text
02464     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02465   g_signal_connect
02466     (window->spin_iterations, "value-changed", window_update, NULL);
02467   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02468   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02469   window->spin_tolerance =
02470     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02471   gtk_widget_set_tooltip_text
02472     (GTK_WIDGET (window->spin_tolerance),
02473      _("Tolerance to set the variable interval on the next iteration"));
02474   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02475   window->spin_bests
02476     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02477   gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->spin_bests),
02479      _("Number of best simulations used to set the variable interval "
02480        "on the next iteration"));
02481   window->label_population
02482     = (GtkLabel *) gtk_label_new (_("Population number"));
02483   window->spin_population
02484     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02485   gtk_widget_set_tooltip_text
02486     (GTK_WIDGET (window->spin_population),
02487      _("Number of population for the genetic algorithm"));
02488   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02489   window->label_generations
02490     = (GtkLabel *) gtk_label_new (_("Generations number"));
02491   window->spin_generations
02492     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02493   gtk_widget_set_tooltip_text
02494     (GTK_WIDGET (window->spin_generations),
02495      _("Number of generations for the genetic algorithm"));
02496   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02497   window->spin_mutation
02498     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02499   gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_mutation),
02501      _("Ratio of mutation for the genetic algorithm"));
02502   window->label_reproduction
02503     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02504   window->spin_reproduction
02505     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02506   gtk_widget_set_tooltip_text
02507     (GTK_WIDGET (window->spin_reproduction),
02508      _("Ratio of reproduction for the genetic algorithm"));
02509   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02510   window->spin_adaptation
02511     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02512   gtk_widget_set_tooltip_text
02513     (GTK_WIDGET (window->spin_adaptation),
02514      _("Ratio of adaptation for the genetic algorithm"));
02515   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02516   window->spin_threshold = (GtkSpinButton *)
02517     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02518                                     precision[DEFAULT_PRECISION]);
02519   gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_threshold),
02521      _("Threshold in the objective function to finish the simulations"));
02522   window->scrolled_threshold = (GtkScrolledWindow *)
02523 #if !GTK4
02524     gtk_scrolled_window_new (NULL, NULL);
02525 #else
02526     gtk_scrolled_window_new ();
02527 #endif
02528   gtk_scrolled_window_set_child (window->scrolled_threshold,
02529                                  GTK_WIDGET (window->spin_threshold));
02530 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02531 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02532 //                         GTK_ALIGN_FILL);
02533
02534   // Creating the hill climbing method properties
02535   window->check_climbing = (GtkCheckButton *)
```

```
02536     gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
02537   g_signal_connect (window->check_climbing, "toggled", window_update, NULL);
02538   window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02539 #if !GTK4
02540   window->button_climbing[0] = (GtkRadioButton *)
02541     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02542 #else
02543   window->button_climbing[0] = (GtkCheckButton *)
02544     gtk_check_button_new_with_mnemonic (label_climbing[0]);
02545 #endif
02546   gtk_grid_attach (window->grid_climbing,
02547                    GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02548   g_signal_connect (window->button_climbing[0], "toggled", window_update, NULL);
02549   for (i = 0; ++i < NCLIMBINGS;)
02550     {
02551 #if !GTK4
02552       window->button_climbing[i] = (GtkRadioButton *)
02553         gtk_radio_button_new_with_mnemonic
02554         (gtk_radio_button_get_group (window->button_climbing[0]),
02555          label_climbing[i]);
02556 #else
02557       window->button_climbing[i] = (GtkCheckButton *)
02558         gtk_check_button_new_with_mnemonic (label_climbing[i]);
02559       gtk_check_button_set_group (window->button_climbing[i],
02560                                   window->button_climbing[0]);
02561 #endif
02562       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02563                                    tip_climbing[i]);
02564       gtk_grid_attach (window->grid_climbing,
02565                        GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02566       g_signal_connect (window->button_climbing[i], "toggled", window_update,
02567                         NULL);
02568     }
02569   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02570   window->spin_steps = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02572   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02573   window->label_estimates
02574     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02575   window->spin_estimates = (GtkSpinButton *)
02576     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02577   window->label_relaxation
02578     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02579   window->spin_relaxation = (GtkSpinButton *)
02580     gtk_spin_button_new_with_range (0., 2., 0.001);
02581   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
02582                    0, NCLIMBINGS, 1, 1);
02583   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
02584                    1, NCLIMBINGS, 1, 1);
02585   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
02586                    0, NCLIMBINGS + 1, 1, 1);
02587   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
02588                    1, NCLIMBINGS + 1, 1, 1);
02589   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
02590                    0, NCLIMBINGS + 2, 1, 1);
02591   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
02592                    1, NCLIMBINGS + 2, 1, 1);
02593
02594   // Creating the array of algorithms
02595   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02596 #if !GTK4
02597   window->button_algorithm[0] = (GtkRadioButton *)
02598     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02599 #else
02600   window->button_algorithm[0] = (GtkCheckButton *)
02601     gtk_check_button_new_with_mnemonic (label_algorithm[0]);
02602 #endif
02603   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02604                                tip_algorithm[0]);
02605   gtk_grid_attach (window->grid_algorithm,
02606                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02607   g_signal_connect (window->button_algorithm[0], "toggled",
02608                     window_set_algorithm, NULL);
02609   for (i = 0; ++i < NALGORITHMS;)
02610     {
02611 #if !GTK4
02612       window->button_algorithm[i] = (GtkRadioButton *)
02613         gtk_radio_button_new_with_mnemonic
02614         (gtk_radio_button_get_group (window->button_algorithm[0]),
02615          label_algorithm[i]);
02616 #else
02617       window->button_algorithm[i] = (GtkCheckButton *)
02618         gtk_check_button_new_with_mnemonic (label_algorithm[i]);
02619       gtk_check_button_set_group (window->button_algorithm[i],
02620                                   window->button_algorithm[0]),
02621 #endif
02622         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
```

```
02623                                   tip_algorithm[i]);
02624       gtk_grid_attach (window->grid_algorithm,
02625                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02626       g_signal_connect (window->button_algorithm[i], "toggled",
02627                         window_set_algorithm, NULL);
02628     }
02629   gtk_grid_attach (window->grid_algorithm,
02630                    GTK_WIDGET (window->label_simulations),
02631                    0, NALGORITHMS, 1, 1);
02632   gtk_grid_attach (window->grid_algorithm,
02633                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02634   gtk_grid_attach (window->grid_algorithm,
02635                    GTK_WIDGET (window->label_iterations),
02636                    0, NALGORITHMS + 1, 1, 1);
02637   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
02638                    1, NALGORITHMS + 1, 1, 1);
02639   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
02640                    0, NALGORITHMS + 2, 1, 1);
02641   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
02642                    1, NALGORITHMS + 2, 1, 1);
02643   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
02644                    0, NALGORITHMS + 3, 1, 1);
02645   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
02646                    1, NALGORITHMS + 3, 1, 1);
02647   gtk_grid_attach (window->grid_algorithm,
02648                    GTK_WIDGET (window->label_population),
02649                    0, NALGORITHMS + 4, 1, 1);
02650   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
02651                    1, NALGORITHMS + 4, 1, 1);
02652   gtk_grid_attach (window->grid_algorithm,
02653                    GTK_WIDGET (window->label_generations),
02654                    0, NALGORITHMS + 5, 1, 1);
02655   gtk_grid_attach (window->grid_algorithm,
02656                    GTK_WIDGET (window->spin_generations),
02657                    1, NALGORITHMS + 5, 1, 1);
02658   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
02659                    0, NALGORITHMS + 6, 1, 1);
02660   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
02661                    1, NALGORITHMS + 6, 1, 1);
02662   gtk_grid_attach (window->grid_algorithm,
02663                    GTK_WIDGET (window->label_reproduction),
02664                    0, NALGORITHMS + 7, 1, 1);
02665   gtk_grid_attach (window->grid_algorithm,
02666                    GTK_WIDGET (window->spin_reproduction),
02667                    1, NALGORITHMS + 7, 1, 1);
02668   gtk_grid_attach (window->grid_algorithm,
02669                    GTK_WIDGET (window->label_adaptation),
02670                    0, NALGORITHMS + 8, 1, 1);
02671   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),
02672                    1, NALGORITHMS + 8, 1, 1);
02673   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
02674                    0, NALGORITHMS + 9, 2, 1);
02675   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
02676                    0, NALGORITHMS + 10, 2, 1);
02677   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
02678                    0, NALGORITHMS + 11, 1, 1);
02679   gtk_grid_attach (window->grid_algorithm,
02680                    GTK_WIDGET (window->scrolled_threshold),
02681                    1, NALGORITHMS + 11, 1, 1);
02682   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02683   gtk_frame_set_child (window->frame_algorithm,
02684                        GTK_WIDGET (window->grid_algorithm));
02685
02686   // Creating the variable widgets
02687   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02688   gtk_widget_set_tooltip_text
02689     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02690   window->id_variable = g_signal_connect
02691     (window->combo_variable, "changed", window_set_variable, NULL);
02692 #if !GTK4
02693   window->button_add_variable = (GtkButton *)
02694     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02695 #else
02696   window->button_add_variable = (GtkButton *)
02697     gtk_button_new_from_icon_name ("list-add");
02698 #endif
02699   g_signal_connect (window->button_add_variable, "clicked", window_add_variable,
02700                     NULL);
02701   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02702                                _("Add variable"));
02703 #if !GTK4
02704   window->button_remove_variable = (GtkButton *)
02705     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02706 #else
02707   window->button_remove_variable = (GtkButton *)
02708     gtk_button_new_from_icon_name ("list-remove");
02709 #endif
```

```
02710   g_signal_connect (window->button_remove_variable, "clicked",
02711                     window_remove_variable, NULL);
02712   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02713                                _("Remove variable"));
02714   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02715   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02716   gtk_widget_set_tooltip_text
02717     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02718   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02719   window->id_variable_label = g_signal_connect
02720     (window->entry_variable, "changed", window_label_variable, NULL);
02721   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02722   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02723     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02724   gtk_widget_set_tooltip_text
02725     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02726   window->scrolled_min = (GtkScrolledWindow *)
02727 #if !GTK4
02728     gtk_scrolled_window_new (NULL, NULL);
02729 #else
02730     gtk_scrolled_window_new ();
02731 #endif
02732   gtk_scrolled_window_set_child (window->scrolled_min,
02733                                  GTK_WIDGET (window->spin_min));
02734   g_signal_connect (window->spin_min, "value-changed",
02735                     window_rangemin_variable, NULL);
02736   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02737   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02738     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02739   gtk_widget_set_tooltip_text
02740     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02741   window->scrolled_max = (GtkScrolledWindow *)
02742 #if !GTK4
02743     gtk_scrolled_window_new (NULL, NULL);
02744 #else
02745     gtk_scrolled_window_new ();
02746 #endif
02747   gtk_scrolled_window_set_child (window->scrolled_max,
02748                                  GTK_WIDGET (window->spin_max));
02749   g_signal_connect (window->spin_max, "value-changed",
02750                     window_rangemax_variable, NULL);
02751   window->check_minabs = (GtkCheckButton *)
02752     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02753   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02754   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02755     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02756   gtk_widget_set_tooltip_text
02757     (GTK_WIDGET (window->spin_minabs),
02758      _("Minimum allowed value of the variable"));
02759   window->scrolled_minabs = (GtkScrolledWindow *)
02760 #if !GTK4
02761     gtk_scrolled_window_new (NULL, NULL);
02762 #else
02763     gtk_scrolled_window_new ();
02764 #endif
02765   gtk_scrolled_window_set_child (window->scrolled_minabs,
02766                                  GTK_WIDGET (window->spin_minabs));
02767   g_signal_connect (window->spin_minabs, "value-changed",
02768                     window_rangeminabs_variable, NULL);
02769   window->check_maxabs = (GtkCheckButton *)
02770     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02771   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02772   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02773     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02774   gtk_widget_set_tooltip_text
02775     (GTK_WIDGET (window->spin_maxabs),
02776      _("Maximum allowed value of the variable"));
02777   window->scrolled_maxabs = (GtkScrolledWindow *)
02778 #if !GTK4
02779     gtk_scrolled_window_new (NULL, NULL);
02780 #else
02781     gtk_scrolled_window_new ();
02782 #endif
02783   gtk_scrolled_window_set_child (window->scrolled_maxabs,
02784                                  GTK_WIDGET (window->spin_maxabs));
02785   g_signal_connect (window->spin_maxabs, "value-changed",
02786                     window_rangemaxabs_variable, NULL);
02787   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02788   window->spin_precision = (GtkSpinButton *)
02789     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02790   gtk_widget_set_tooltip_text
02791     (GTK_WIDGET (window->spin_precision),
02792      _("Number of precision floating point digits\n"
02793        "0 is for integer numbers"));
02794   g_signal_connect (window->spin_precision, "value-changed",
02795                     window_precision_variable, NULL);
02796   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
```

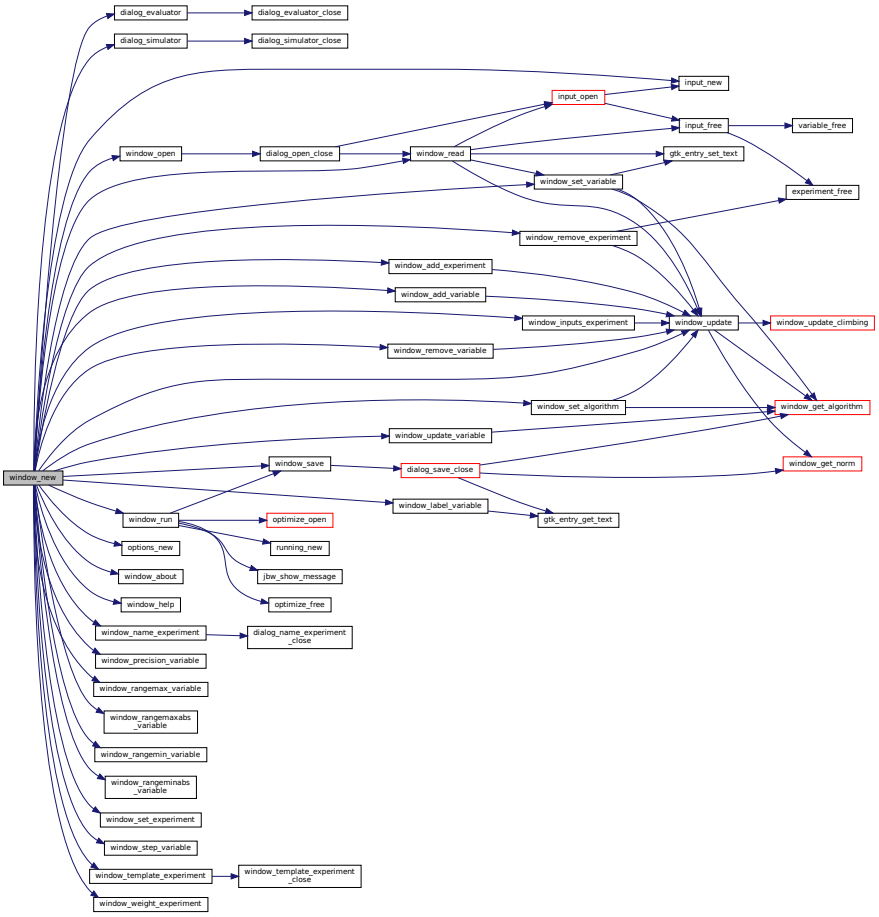
```
02797   window->spin_sweeps =
02798     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02799   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02800                                _("Number of steps sweeping the variable"));
02801   g_signal_connect (window->spin_sweeps, "value-changed",
02802                     window_update_variable, NULL);
02803   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02804   window->spin_bits
02805     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02806   gtk_widget_set_tooltip_text
02807     (GTK_WIDGET (window->spin_bits),
02808      _("Number of bits to encode the variable"));
02809   g_signal_connect
02810     (window->spin_bits, "value-changed", window_update_variable, NULL);
02811   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02812   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02813     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02814   gtk_widget_set_tooltip_text
02815     (GTK_WIDGET (window->spin_step),
02816      _("Initial step size for the hill climbing method"));
02817   window->scrolled_step = (GtkScrolledWindow *)
02818 #if !GTK4
02819     gtk_scrolled_window_new (NULL, NULL);
02820 #else
02821     gtk_scrolled_window_new ();
02822 #endif
02823   gtk_scrolled_window_set_child (window->scrolled_step,
02824                                  GTK_WIDGET (window->spin_step));
02825   g_signal_connect
02826     (window->spin_step, "value-changed", window_step_variable, NULL);
02827   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02828   gtk_grid_attach (window->grid_variable,
02829                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02830   gtk_grid_attach (window->grid_variable,
02831                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02832   gtk_grid_attach (window->grid_variable,
02833                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02834   gtk_grid_attach (window->grid_variable,
02835                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02836   gtk_grid_attach (window->grid_variable,
02837                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02838   gtk_grid_attach (window->grid_variable,
02839                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02840   gtk_grid_attach (window->grid_variable,
02841                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02842   gtk_grid_attach (window->grid_variable,
02843                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02844   gtk_grid_attach (window->grid_variable,
02845                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02846   gtk_grid_attach (window->grid_variable,
02847                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02848   gtk_grid_attach (window->grid_variable,
02849                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02850   gtk_grid_attach (window->grid_variable,
02851                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02852   gtk_grid_attach (window->grid_variable,
02853                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02854   gtk_grid_attach (window->grid_variable,
02855                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02856   gtk_grid_attach (window->grid_variable,
02857                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02858   gtk_grid_attach (window->grid_variable,
02859                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02860   gtk_grid_attach (window->grid_variable,
02861                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02862   gtk_grid_attach (window->grid_variable,
02863                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02864   gtk_grid_attach (window->grid_variable,
02865                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02866   gtk_grid_attach (window->grid_variable,
02867                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02868   gtk_grid_attach (window->grid_variable,
02869                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02870   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02871   gtk_frame_set_child (window->frame_variable,
02872                        GTK_WIDGET (window->grid_variable));
02873
02874   // Creating the experiment widgets
02875   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02876   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02877                                _("Experiment selector"));
02878   window->id_experiment = g_signal_connect
02879     (window->combo_experiment, "changed", window_set_experiment, NULL);
02880 #if !GTK4
02881   window->button_add_experiment = (GtkButton *)
02882     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02883 #else
```

```
02884   window->button_add_experiment = (GtkButton *)
02885     gtk_button_new_from_icon_name ("list-add");
02886 #endif
02887   g_signal_connect
02888     (window->button_add_experiment, "clicked", window_add_experiment, NULL);
02889   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02890                                _("Add experiment"));
02891 #if !GTK4
02892   window->button_remove_experiment = (GtkButton *)
02893     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02894 #else
02895   window->button_remove_experiment = (GtkButton *)
02896     gtk_button_new_from_icon_name ("list-remove");
02897 #endif
02898   g_signal_connect (window->button_remove_experiment, "clicked",
02899                     window_remove_experiment, NULL);
02900   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02901                                _("Remove experiment"));
02902   window->label_experiment
02903     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02904   window->button_experiment = (GtkButton *)
02905     gtk_button_new_with_mnemonic (_("Experimental data file"));
02906   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02907                                _("Experimental data file"));
02908   g_signal_connect (window->button_experiment, "clicked",
02909                     window_name_experiment, NULL);
02910   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02911   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02912   window->spin_weight
02913     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02914   gtk_widget_set_tooltip_text
02915     (GTK_WIDGET (window->spin_weight),
02916      _("Weight factor to build the objective function"));
02917   g_signal_connect
02918     (window->spin_weight, "value-changed", window_weight_experiment, NULL);
02919   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02920   gtk_grid_attach (window->grid_experiment,
02921                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02922   gtk_grid_attach (window->grid_experiment,
02923                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02924   gtk_grid_attach (window->grid_experiment,
02925                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02926   gtk_grid_attach (window->grid_experiment,
02927                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02928   gtk_grid_attach (window->grid_experiment,
02929                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02930   gtk_grid_attach (window->grid_experiment,
02931                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02932   gtk_grid_attach (window->grid_experiment,
02933                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02934   for (i = 0; i < MAX_NINPUTS; ++i)
02935     {
02936       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02937       window->check_template[i] = (GtkCheckButton *)
02938         gtk_check_button_new_with_label (buffer3);
02939       window->id_template[i]
02940         = g_signal_connect (window->check_template[i], "toggled",
02941                             window_inputs_experiment, NULL);
02942       gtk_grid_attach (window->grid_experiment,
02943                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02944       window->button_template[i] = (GtkButton *)
02945         gtk_button_new_with_mnemonic (_("Input template"));
02946
02947       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02948                                    _("Experimental input template file"));
02949       window->id_input[i] =
02950         g_signal_connect_swapped (window->button_template[i], "clicked",
02951                                   (GCallback) window_template_experiment,
02952                                   (void *) (size_t) i);
02953       gtk_grid_attach (window->grid_experiment,
02954                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02955     }
02956   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02957   gtk_frame_set_child (window->frame_experiment,
02958                        GTK_WIDGET (window->grid_experiment));
02959
02960   // Creating the error norm widgets
02961   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02962   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02963   gtk_frame_set_child (window->frame_norm, GTK_WIDGET (window->grid_norm));
02964 #if !GTK4
02965   window->button_norm[0] = (GtkRadioButton *)
02966     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02967 #else
02968   window->button_norm[0] = (GtkCheckButton *)
02969     gtk_check_button_new_with_mnemonic (label_norm[0]);
02970 #endif
```

```
02971   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02972                                tip_norm[0]);
02973   gtk_grid_attach (window->grid_norm,
02974                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02975   g_signal_connect (window->button_norm[0], "toggled", window_update, NULL);
02976   for (i = 0; ++i < NNORMS;)
02977     {
02978 #if !GTK4
02979       window->button_norm[i] = (GtkRadioButton *)
02980         gtk_radio_button_new_with_mnemonic
02981         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02982 #else
02983       window->button_norm[i] = (GtkCheckButton *)
02984         gtk_check_button_new_with_mnemonic (label_norm[i]);
02985       gtk_check_button_set_group (window->button_norm[i],
02986                                   window->button_norm[0]);
02987 #endif
02988       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02989                                    tip_norm[i]);
02990       gtk_grid_attach (window->grid_norm,
02991                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02992       g_signal_connect (window->button_norm[i], "toggled", window_update, NULL);
02993     }
02994   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02995   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02996   window->spin_p = (GtkSpinButton *)
02997     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02998   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02999                                _("P parameter for the P error norm"));
03000   window->scrolled_p = (GtkScrolledWindow *)
03001 #if !GTK4
03002     gtk_scrolled_window_new (NULL, NULL);
03003 #else
03004     gtk_scrolled_window_new ();
03005 #endif
03006   gtk_scrolled_window_set_child (window->scrolled_p,
03007                                  GTK_WIDGET (window->spin_p));
03008   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
03009   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
03010   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
03011                    1, 2, 1, 2);
03012
03013   // Creating the grid and attaching the widgets to the grid
03014   window->grid = (GtkGrid *) gtk_grid_new ();
03015   gtk_grid_attach (window->grid, GTK_WIDGET (window->box_buttons), 0, 0, 3, 1);
03016   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
03017   gtk_grid_attach (window->grid,
03018                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
03019   gtk_grid_attach (window->grid,
03020                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
03021   gtk_grid_attach (window->grid,
03022                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
03023   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
03024   gtk_window_set_child (window->window, GTK_WIDGET (window->grid));
03025
03026   // Setting the window logo
03027   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03028 #if !GTK4
03029   gtk_window_set_icon (window->window, window->logo);
03030 #endif
03031
03032   // Showing the window
03033 #if !GTK4
03034   gtk_widget_show_all (GTK_WIDGET (window->window));
03035 #else
03036   gtk_widget_show (GTK_WIDGET (window->window));
03037 #endif
03038
03039   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
03040 #if GTK_MINOR_VERSION >= 16
03041   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03042   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03043   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03044   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03045   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
03046   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
03047   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
03048 #endif
03049
03050   // Reading initial example
03051   input_new ();
03052   buffer2 = g_get_current_dir ();
03053   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03054   g_free (buffer2);
03055   window_read (buffer);
03056   g_free (buffer);
03057
```

```
03058 #if DEBUG_INTERFACE
03059   fprintf (stderr, "window_new:  start\n");
03060 #endif
03061 }
```

Here is the call graph for this function:



4.11.3.27 window_open()

```
static void window_open ( )  [static]
```

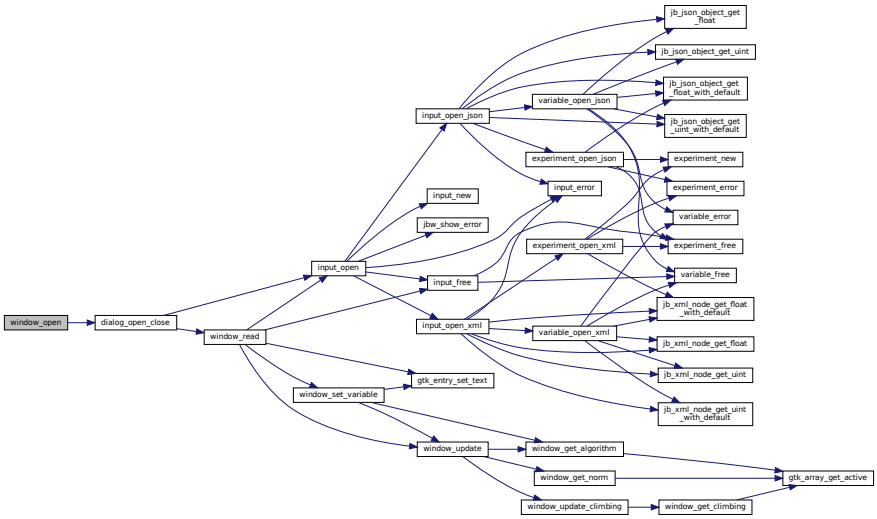
Function to open the input data.

Definition at line 2118 of file interface.c.

```
02119 {
02120   GtkFileChooserDialog *dlg;
02121   GtkFileFilter *filter;
02122
02123 #if DEBUG_INTERFACE
02124   fprintf (stderr, "window_open:  start\n");
02125 #endif
02126
02127   // Opening dialog
02128   dlg = (GtkFileChooserDialog *)
02129     gtk_file_chooser_dialog_new (_("Open input file"),
02130                                  window->window,
02131                                  GTK_FILE_CHOOSER_ACTION_OPEN,
```

```
02132                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
02133                                 _("_OK"), GTK_RESPONSE_OK, NULL);
02134
02135   // Adding XML filter
02136   filter = (GtkFileFilter *) gtk_file_filter_new ();
02137   gtk_file_filter_set_name (filter, "XML");
02138   gtk_file_filter_add_pattern (filter, "*.xml");
02139   gtk_file_filter_add_pattern (filter, "*.XML");
02140   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02141
02142   // Adding JSON filter
02143   filter = (GtkFileFilter *) gtk_file_filter_new ();
02144   gtk_file_filter_set_name (filter, "JSON");
02145   gtk_file_filter_add_pattern (filter, "*.json");
02146   gtk_file_filter_add_pattern (filter, "*.JSON");
02147   gtk_file_filter_add_pattern (filter, "*.js");
02148   gtk_file_filter_add_pattern (filter, "*.JS");
02149   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02150
02151   // Connecting the close function
02152   g_signal_connect (dlg, "response", G_CALLBACK (dialog_open_close), NULL);
02153
02154   // Showing modal dialog
02155   gtk_window_present (GTK_WINDOW (dlg));
02156
02157 #if DEBUG_INTERFACE
02158   fprintf (stderr, "window_open:  end\n");
02159 #endif
02160 }
```

Here is the call graph for this function:



4.11.3.28 window_precision_variable()

```
static void window_precision_variable ( )  [static]
```

Function to update the variable precision in the main window.

Definition at line 1810 of file interface.c.

```
01811 {
01812   unsigned int i;
01813 #if DEBUG_INTERFACE
01814   fprintf (stderr, "window_precision_variable:  start\n");
01815 #endif
01816   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
```

```
01817   input->variable[i].precision
01818     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01819   gtk_spin_button_set_digits (window->spin_min, input->variable[i].precision);
01820   gtk_spin_button_set_digits (window->spin_max, input->variable[i].precision);
01821   gtk_spin_button_set_digits (window->spin_minabs,
01822                               input->variable[i].precision);
01823   gtk_spin_button_set_digits (window->spin_maxabs,
01824                               input->variable[i].precision);
01825 #if DEBUG_INTERFACE
01826   fprintf (stderr, "window_precision_variable:  end\n");
01827 #endif
01828 }
```

4.11.3.29 window_rangemax_variable()

```
static void window_rangemax_variable ( )  [static]
```

Function to update the variable rangemax in the main window.

Definition at line 1851 of file interface.c.

```
01852 {
01853   unsigned int i;
01854 #if DEBUG_INTERFACE
01855   fprintf (stderr, "window_rangemax_variable:  start\n");
01856 #endif
01857   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01858   input->variable[i].rangemax = gtk_spin_button_get_value (window->spin_max);
01859 #if DEBUG_INTERFACE
01860   fprintf (stderr, "window_rangemax_variable:  end\n");
01861 #endif
01862 }
```

4.11.3.30 window_rangemaxabs_variable()

```
static void window_rangemaxabs_variable ( )  [static]
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1886 of file interface.c.

```
01887 {
01888   unsigned int i;
01889 #if DEBUG_INTERFACE
01890   fprintf (stderr, "window_rangemaxabs_variable:  start\n");
01891 #endif
01892   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01893   input->variable[i].rangemaxabs
01894     = gtk_spin_button_get_value (window->spin_maxabs);
01895 #if DEBUG_INTERFACE
01896   fprintf (stderr, "window_rangemaxabs_variable:  end\n");
01897 #endif
01898 }
```

4.11.3.31 window_rangemin_variable()

```
static void window_rangemin_variable ( )  [static]
```

Function to update the variable rangemin in the main window.

Definition at line 1834 of file interface.c.

```
01835 {
01836   unsigned int i;
01837 #if DEBUG_INTERFACE
01838   fprintf (stderr, "window_rangemin_variable:  start\n");
01839 #endif
01840   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01841   input->variable[i].rangemin = gtk_spin_button_get_value (window->spin_min);
01842 #if DEBUG_INTERFACE
01843   fprintf (stderr, "window_rangemin_variable:  end\n");
01844 #endif
01845 }
```

4.11.3.32 window_rangeminabs_variable()

```
static void window_rangeminabs_variable ( )  [static]
```

Function to update the variable rangeminabs in the main window.

Definition at line 1868 of file interface.c.

```
01869 {
01870   unsigned int i;
01871 #if DEBUG_INTERFACE
01872   fprintf (stderr, "window_rangeminabs_variable:  start\n");
01873 #endif
01874   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01875   input->variable[i].rangeminabs
01876     = gtk_spin_button_get_value (window->spin_minabs);
01877 #if DEBUG_INTERFACE
01878   fprintf (stderr, "window_rangeminabs_variable:  end\n");
01879 #endif
01880 }
```

4.11.3.33 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

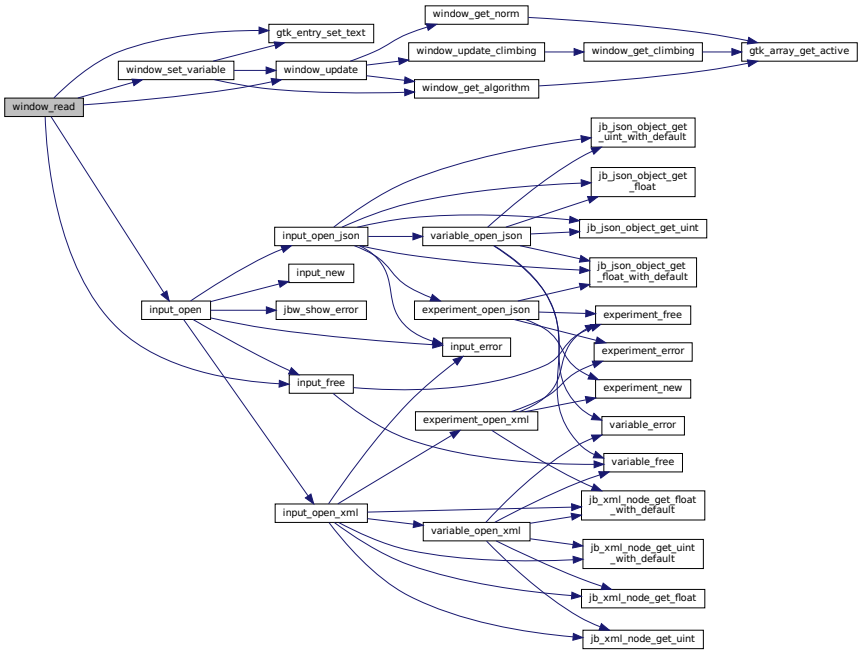
filename	File name.

Definition at line 1960 of file interface.c.

```
01961 {
01962   unsigned int i;
01963 #if DEBUG_INTERFACE
01964   fprintf (stderr, "window_read:  start\n");
01965 #endif
01966
01967   // Reading new input file
01968   input_free ();
01969   input->result = input->variables = NULL;
01970   if (!input_open (filename))
01971     {
01972 #if DEBUG_INTERFACE
01973       fprintf (stderr, "window_read:  end\n");
01974 #endif
01975       return 0;
01976     }
01977
01978   // Setting GTK+ widgets data
01979   gtk_entry_set_text (window->entry_result, input->result);
01980   gtk_entry_set_text (window->entry_variables, input->variables);
01981   gtk_button_set_label (window->button_simulator, input->simulator);
01982   gtk_check_button_set_active (window->check_evaluator,
01983                                (size_t) input->evaluator);
01984   if (input->evaluator)
01985     gtk_button_set_label (window->button_evaluator, input->evaluator);
01986   gtk_check_button_set_active (window->button_algorithm[input->algorithm],
01987                                TRUE);
01988   switch (input->algorithm)
01989     {
```

```
01990     case ALGORITHM_MONTE_CARLO:
01991       gtk_spin_button_set_value (window->spin_simulations,
01992                                  (gdouble) input->nsimulations);
01993       // fallthrough
01994     case ALGORITHM_SWEEP:
01995     case ALGORITHM_ORTHOGONAL:
01996       gtk_spin_button_set_value (window->spin_iterations,
01997                                  (gdouble) input->niterations);
01998       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
01999       gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
02000       gtk_check_button_set_active (window->check_climbing, input->nsteps);
02001       if (input->nsteps)
02002         {
02003           gtk_check_button_set_active
02004             (window->button_climbing[input->climbing], TRUE);
02005           gtk_spin_button_set_value (window->spin_steps,
02006                                      (gdouble) input->nsteps);
02007           gtk_spin_button_set_value (window->spin_relaxation,
02008                                      (gdouble) input->relaxation);
02009           switch (input->climbing)
02010             {
02011             case CLIMBING_METHOD_RANDOM:
02012               gtk_spin_button_set_value (window->spin_estimates,
02013                                          (gdouble) input->nestimates);
02014             }
02015         }
02016       break;
02017     default:
02018       gtk_spin_button_set_value (window->spin_population,
02019                                  (gdouble) input->nsimulations);
02020       gtk_spin_button_set_value (window->spin_generations,
02021                                  (gdouble) input->niterations);
02022       gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
02023       gtk_spin_button_set_value (window->spin_reproduction,
02024                                  input->reproduction_ratio);
02025       gtk_spin_button_set_value (window->spin_adaptation,
02026                                  input->adaptation_ratio);
02027     }
02028   gtk_check_button_set_active (window->button_norm[input->norm], TRUE);
02029   gtk_spin_button_set_value (window->spin_p, input->p);
02030   gtk_spin_button_set_value (window->spin_threshold, input->threshold);
02031   g_signal_handler_block (window->combo_experiment, window->id_experiment);
02032   gtk_combo_box_text_remove_all (window->combo_experiment);
02033   for (i = 0; i < input->nexperiments; ++i)
02034     gtk_combo_box_text_append_text (window->combo_experiment,
02035                                     input->experiment[i].name);
02036   g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
02037   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
02038   g_signal_handler_block (window->combo_variable, window->id_variable);
02039   g_signal_handler_block (window->entry_variable, window->id_variable_label);
02040   gtk_combo_box_text_remove_all (window->combo_variable);
02041   for (i = 0; i < input->nvariables; ++i)
02042     gtk_combo_box_text_append_text (window->combo_variable,
02043                                     input->variable[i].name);
02044   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
02045   g_signal_handler_unblock (window->combo_variable, window->id_variable);
02046   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02047   window_set_variable ();
02048   window_update ();
02049
02050 #if DEBUG_INTERFACE
02051   fprintf (stderr, "window_read:  end\n");
02052 #endif
02053   return 1;
02054 }
```

Here is the call graph for this function:



4.11.3.34 window_remove_experiment()

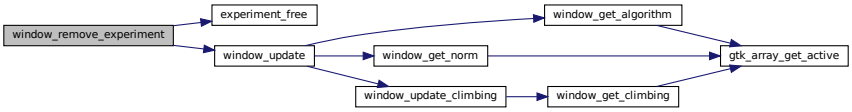
```
static void window_remove_experiment ( )  [static]
```

Function to remove an experiment in the main window.

Definition at line 1384 of file interface.c.

```
01385 {
01386   unsigned int i, j;
01387 #if DEBUG_INTERFACE
01388   fprintf (stderr, "window_remove_experiment:  start\n");
01389 #endif
01390   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01391   g_signal_handler_block (window->combo_experiment, window->id_experiment);
01392   gtk_combo_box_text_remove (window->combo_experiment, i);
01393   g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
01394   experiment_free (input->experiment + i, input->type);
01395   --input->nexperiments;
01396   for (j = i; j < input->nexperiments; ++j)
01397     memcpy (input->experiment + j, input->experiment + j + 1,
01398             sizeof (Experiment));
01399   j = input->nexperiments - 1;
01400   if (i > j)
01401     i = j;
01402   for (j = 0; j < input->experiment->ninputs; ++j)
01403     g_signal_handler_block (window->button_template[j], window->id_input[j]);
01404   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01405   for (j = 0; j < input->experiment->ninputs; ++j)
01406     g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
01407   window_update ();
01408 #if DEBUG_INTERFACE
01409   fprintf (stderr, "window_remove_experiment:  end\n");
01410 #endif
01411 }
```

Here is the call graph for this function:



4.11.3.35 window_remove_variable()

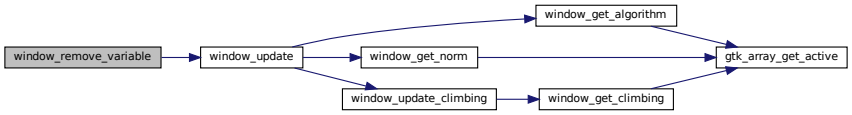
```
static void window_remove_variable ( )  [static]
```

Function to remove a variable in the main window.

Definition at line 1722 of file interface.c.

```
01723 {
01724   unsigned int i, j;
01725 #if DEBUG_INTERFACE
01726   fprintf (stderr, "window_remove_variable:  start\n");
01727 #endif
01728   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01729   g_signal_handler_block (window->combo_variable, window->id_variable);
01730   gtk_combo_box_text_remove (window->combo_variable, i);
01731   g_signal_handler_unblock (window->combo_variable, window->id_variable);
01732   xmlFree (input->variable[i].name);
01733   --input->nvariables;
01734   for (j = i; j < input->nvariables; ++j)
01735     memcpy (input->variable + j, input->variable + j + 1, sizeof (Variable));
01736   j = input->nvariables - 1;
01737   if (i > j)
01738     i = j;
01739   g_signal_handler_block (window->entry_variable, window->id_variable_label);
01740   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01741   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01742   window_update ();
01743 #if DEBUG_INTERFACE
01744   fprintf (stderr, "window_remove_variable:  end\n");
01745 #endif
01746 }
```

Here is the call graph for this function:



4.11.3.36 window_run()

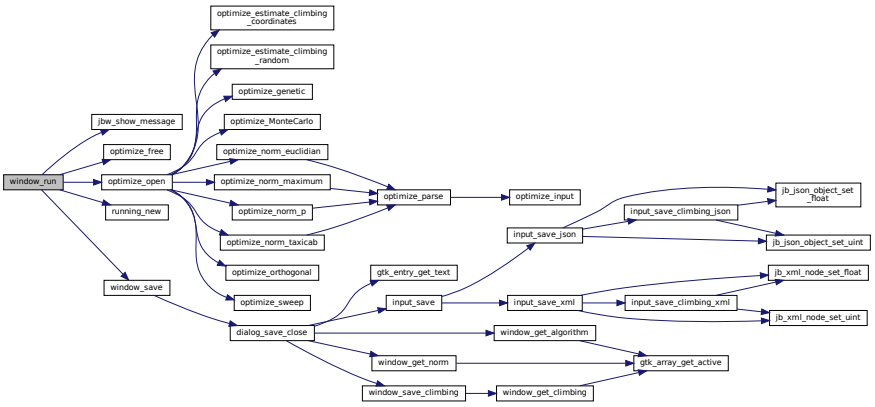
```
static void window_run ( )  [static]
```

Function to run a optimization.

Definition at line 1019 of file interface.c.

```
01020 {
01021   GMainContext *context;
01022   char *msg, *msg2, buffer[64], buffer2[64];
01023   unsigned int i;
01024 #if DEBUG_INTERFACE
01025   fprintf (stderr, "window_run:  start\n");
01026 #endif
01027   window_save ();
01028   running_new ();
01029   context = g_main_context_default ();
01030   while (g_main_context_pending (context))
01031     g_main_context_iteration (context, 0);
01032   optimize_open ();
01033 #if DEBUG_INTERFACE
01034   fprintf (stderr, "window_run:  closing running dialog\n");
01035 #endif
01036   gtk_spinner_stop (running->spinner);
01037   gtk_window_destroy (GTK_WINDOW (running->dialog));
01038 #if DEBUG_INTERFACE
01039   fprintf (stderr, "window_run:  displaying results\n");
01040 #endif
01041   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01042   msg2 = g_strdup (buffer);
01043   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01044     {
01045       snprintf (buffer, 64, "%s = %s\n",
01046                 input->variable[i].name, format[input->variable[i].precision]);
01047       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01048       msg = g_strconcat (msg2, buffer2, NULL);
01049       g_free (msg2);
01050     }
01051   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01052             optimize->calculation_time);
01053   msg = g_strconcat (msg2, buffer, NULL);
01054   g_free (msg2);
01055   jbw_show_message (_("Best result"), msg, INFO_TYPE);
01056   g_free (msg);
01057 #if DEBUG_INTERFACE
01058   fprintf (stderr, "window_run:  freeing memory\n");
01059 #endif
01060   optimize_free ();
01061 #if DEBUG_INTERFACE
01062   fprintf (stderr, "window_run:  end\n");
01063 #endif
01064 }
```

Here is the call graph for this function:



4.11.3.37 window_save()

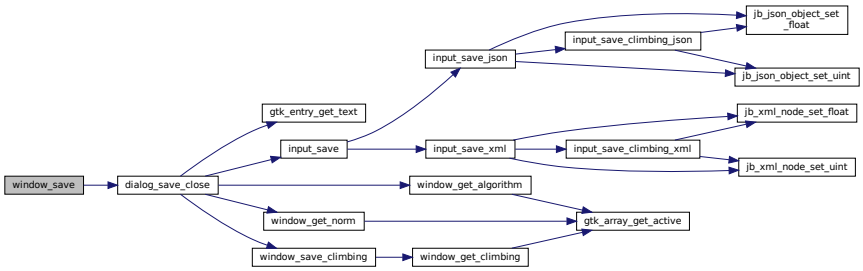
```
static void window_save ( )  [static]
```

Function to save the input file.

Definition at line 959 of file interface.c.

```
00960 {
00961   GtkFileChooserDialog *dlg;
00962   GtkFileFilter *filter1, *filter2;
00963   char *buffer;
00964
00965 #if DEBUG_INTERFACE
00966   fprintf (stderr, "window_save:  start\n");
00967 #endif
00968
00969   // Opening the saving dialog
00970   dlg = (GtkFileChooserDialog *)
00971     gtk_file_chooser_dialog_new (_("Save file"),
00972                                  window->window,
00973                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00974                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00975                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00976 #if !GTK4
00977   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00978 #endif
00979   buffer = g_build_filename (input->directory, input->name, NULL);
00980   gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
00981   g_free (buffer);
00982
00983   // Adding XML filter
00984   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00985   gtk_file_filter_set_name (filter1, "XML");
00986   gtk_file_filter_add_pattern (filter1, "*.xml");
00987   gtk_file_filter_add_pattern (filter1, "*.XML");
00988   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00989
00990   // Adding JSON filter
00991   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00992   gtk_file_filter_set_name (filter2, "JSON");
00993   gtk_file_filter_add_pattern (filter2, "*.json");
00994   gtk_file_filter_add_pattern (filter2, "*.JSON");
00995   gtk_file_filter_add_pattern (filter2, "*.js");
00996   gtk_file_filter_add_pattern (filter2, "*.JS");
00997   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00998
00999   if (input->type == INPUT_TYPE_XML)
01000     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
01001   else
01002     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
01003
01004   // Connecting the close function
01005   g_signal_connect (dlg, "response", G_CALLBACK (dialog_save_close), NULL);
01006
01007   // Showing modal dialog
01008   gtk_window_present (GTK_WINDOW (dlg));
01009
01010 #if DEBUG_INTERFACE
01011   fprintf (stderr, "window_save:  end\n");
01012 #endif
01013 }
```

Here is the call graph for this function:



4.11.3.38 window_save_climbing()

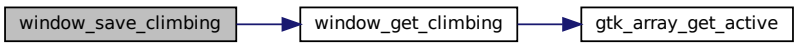
```
static void window_save_climbing ( )  [static]
```

Function to save the hill climbing method data in the input file.

Definition at line 821 of file interface.c.

```
00822 {
00823 #if DEBUG_INTERFACE
00824   fprintf (stderr, "window_save_climbing:  start\n");
00825 #endif
00826   if (gtk_check_button_get_active (window->check_climbing))
00827     {
00828       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
00829       input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
00830       switch (window_get_climbing ())
00831         {
00832         case CLIMBING_METHOD_COORDINATES:
00833           input->climbing = CLIMBING_METHOD_COORDINATES;
00834           break;
00835         default:
00836           input->climbing = CLIMBING_METHOD_RANDOM;
00837           input->nestimates
00838             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00839         }
00840     }
00841   else
00842     input->nsteps = 0;
00843 #if DEBUG_INTERFACE
00844   fprintf (stderr, "window_save_climbing:  end\n");
00845 #endif
00846 }
```

Here is the call graph for this function:



4.11.3.39 window_set_algorithm()

```
static void window_set_algorithm ( )  [static]
```

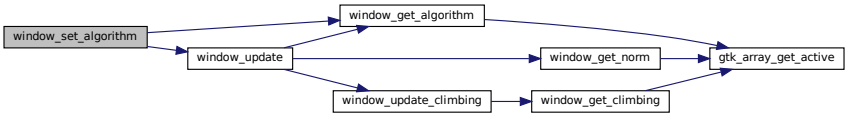
Function to avoid memory errors changing the algorithm.

Definition at line 1321 of file interface.c.

```
01322 {
01323   int i;
01324 #if DEBUG_INTERFACE
01325   fprintf (stderr, "window_set_algorithm:  start\n");
01326 #endif
01327   i = window_get_algorithm ();
01328   switch (i)
01329     {
01330     case ALGORITHM_SWEEP:
01331     case ALGORITHM_ORTHOGONAL:
01332       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01333       if (i < 0)
01334         i = 0;
01335       gtk_spin_button_set_value (window->spin_sweeps,
01336                                  (gdouble) input->variable[i].nsweeps);
01337       break;
01338     case ALGORITHM_GENETIC:
01339       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
```

```
01340       if (i < 0)
01341         i = 0;
01342       gtk_spin_button_set_value (window->spin_bits,
01343                                  (gdouble) input->variable[i].nbits);
01344     }
01345   window_update ();
01346 #if DEBUG_INTERFACE
01347   fprintf (stderr, "window_set_algorithm:  end\n");
01348 #endif
01349 }
```

Here is the call graph for this function:



4.11.3.40 window_set_experiment()

```
static void window_set_experiment ( )  [static]
```

Function to set the experiment data in the main window.

Definition at line 1355 of file interface.c.

```
01356 {
01357   unsigned int i, j;
01358   char *buffer1;
01359 #if DEBUG_INTERFACE
01360   fprintf (stderr, "window_set_experiment:  start\n");
01361 #endif
01362   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01363   gtk_spin_button_set_value (window->spin_weight, input->experiment[i].weight);
01364   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01365   gtk_button_set_label (window->button_experiment, buffer1);
01366   g_free (buffer1);
01367   for (j = 0; j < input->experiment->ninputs; ++j)
01368     {
01369       g_signal_handler_block (window->button_template[j], window->id_input[j]);
01370       gtk_button_set_label (window->button_template[j],
01371                             input->experiment[i].stencil[j]);
01372       g_signal_handler_unblock
01373         (window->button_template[j], window->id_input[j]);
01374     }
01375 #if DEBUG_INTERFACE
01376   fprintf (stderr, "window_set_experiment:  end\n");
01377 #endif
01378 }
```

4.11.3.41 window_set_variable()

```
static void window_set_variable ( )  [static]
```

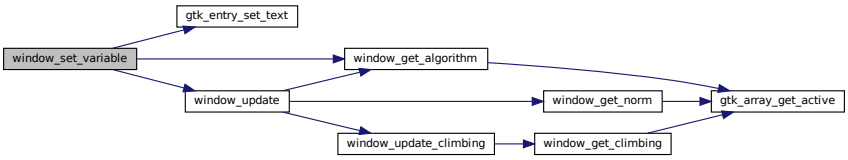
Function to set the variable data in the main window.

Definition at line 1649 of file interface.c.

```
01650 {
01651   unsigned int i;
```

```
01652 #if DEBUG_INTERFACE
01653   fprintf (stderr, "window_set_variable:  start\n");
01654 #endif
01655   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01656   g_signal_handler_block (window->entry_variable, window->id_variable_label);
01657   gtk_entry_set_text (window->entry_variable, input->variable[i].name);
01658   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01659   gtk_spin_button_set_value (window->spin_min, input->variable[i].rangemin);
01660   gtk_spin_button_set_value (window->spin_max, input->variable[i].rangemax);
01661   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01662     {
01663       gtk_spin_button_set_value (window->spin_minabs,
01664                                  input->variable[i].rangeminabs);
01665       gtk_check_button_set_active (window->check_minabs, 1);
01666     }
01667   else
01668     {
01669       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01670       gtk_check_button_set_active (window->check_minabs, 0);
01671     }
01672   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01673     {
01674       gtk_spin_button_set_value (window->spin_maxabs,
01675                                  input->variable[i].rangemaxabs);
01676       gtk_check_button_set_active (window->check_maxabs, 1);
01677     }
01678   else
01679     {
01680       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01681       gtk_check_button_set_active (window->check_maxabs, 0);
01682     }
01683   gtk_spin_button_set_value (window->spin_precision,
01684                              input->variable[i].precision);
01685   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
01686   if (input->nsteps)
01687     gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
01688 #if DEBUG_INTERFACE
01689   fprintf (stderr, "window_set_variable:  precision[%u]=%u\n", i,
01690            input->variable[i].precision);
01691 #endif
01692   switch (window_get_algorithm ())
01693     {
01694     case ALGORITHM_SWEEP:
01695     case ALGORITHM_ORTHOGONAL:
01696       gtk_spin_button_set_value (window->spin_sweeps,
01697                                  (gdouble) input->variable[i].nsweeps);
01698 #if DEBUG_INTERFACE
01699       fprintf (stderr, "window_set_variable:  nsweeps[%u]=%u\n", i,
01700                input->variable[i].nsweeps);
01701 #endif
01702       break;
01703     case ALGORITHM_GENETIC:
01704       gtk_spin_button_set_value (window->spin_bits,
01705                                  (gdouble) input->variable[i].nbits);
01706 #if DEBUG_INTERFACE
01707       fprintf (stderr, "window_set_variable:  nbits[%u]=%u\n", i,
01708                input->variable[i].nbits);
01709 #endif
01710       break;
01711     }
01712   window_update ();
01713 #if DEBUG_INTERFACE
01714   fprintf (stderr, "window_set_variable:  end\n");
01715 #endif
01716 }
```

Here is the call graph for this function:



4.11.3.42 window_step_variable()

```
static void window_step_variable ( )  [static]
```

Function to update the variable step in the main window.

Definition at line 1904 of file interface.c.

```
01905 {
01906   unsigned int i;
01907 #if DEBUG_INTERFACE
01908   fprintf (stderr, "window_step_variable:  start\n");
01909 #endif
01910   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01911   input->variable[i].step = gtk_spin_button_get_value (window->spin_step);
01912 #if DEBUG_INTERFACE
01913   fprintf (stderr, "window_step_variable:  end\n");
01914 #endif
01915 }
```

4.11.3.43 window_template_experiment()

```
static void window_template_experiment (
             void * data )  [static]
```

Function to update the experiment i-th input template in the main window.

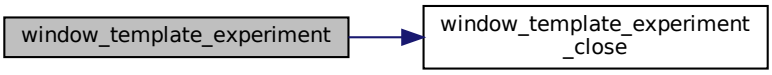
Parameters

data	Callback data (i-th input template).

Definition at line 1612 of file interface.c.

```
01614 {
01615   GtkFileChooserDialog *dlg;
01616   GMainLoop *loop;
01617   const char *buffer;
01618   unsigned int i;
01619 #if DEBUG_INTERFACE
01620   fprintf (stderr, "window_template_experiment:  start\n");
01621 #endif
01622   i = (size_t) data;
01623   buffer = gtk_button_get_label (window->button_template[i]);
01624   dlg = (GtkFileChooserDialog *)
01625     gtk_file_chooser_dialog_new (_("Open template file"),
01626                                  window->window,
01627                                  GTK_FILE_CHOOSER_ACTION_OPEN,
01628                                  _("_Cancel"),
01629                                  GTK_RESPONSE_CANCEL,
01630                                  _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
01631   gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01632   g_signal_connect (dlg, "response",
01633                     G_CALLBACK (window_template_experiment_close), data);
01634   gtk_window_present (GTK_WINDOW (dlg));
01635   loop = g_main_loop_new (NULL, 0);
01636   g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01637                             loop);
01638   g_main_loop_run (loop);
01639   g_main_loop_unref (loop);
01640 #if DEBUG_INTERFACE
01641   fprintf (stderr, "window_template_experiment:  end\n");
01642 #endif
01643 }
```

Here is the call graph for this function:



4.11.3.44 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

dlg	GtkFileChooserDialg struct.
response↩ _id	Response identifier.
data	Function data.

Definition at line 1572 of file interface.c.

```
01577 {
01578   GFile *file1, *file2;
01579   char *buffer1, *buffer2;
01580   unsigned int i, j;
01581 #if DEBUG_INTERFACE
01582   fprintf (stderr, "window_template_experiment_close:  start\n");
01583 #endif
01584   if (response_id == GTK_RESPONSE_OK)
01585     {
01586       i = (size_t) data;
01587       j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01588       buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01589       file1 = g_file_new_for_path (buffer1);
01590       file2 = g_file_new_for_path (input->directory);
01591       buffer2 = g_file_get_relative_path (file2, file1);
01592       if (input->type == INPUT_TYPE_XML)
01593         input->experiment[j].stencil[i]
01594           = (char *) xmlStrdup ((xmlChar *) buffer2);
01595       else
01596         input->experiment[j].stencil[i] = g_strdup (buffer2);
01597       g_free (buffer2);
01598       g_object_unref (file2);
01599       g_object_unref (file1);
01600       g_free (buffer1);
01601     }
01602   gtk_window_destroy (GTK_WINDOW (dlg));
01603 #if DEBUG_INTERFACE
01604   fprintf (stderr, "window_template_experiment_close:  end\n");
01605 #endif
01606 }
```

4.11.3.45 window_update()

```
static void window_update ( )  [static]
```

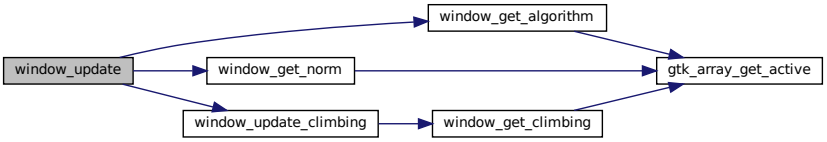
Function to update the main window view.

Definition at line 1169 of file interface.c.

```
01170 {
01171   unsigned int i;
01172 #if DEBUG_INTERFACE
01173   fprintf (stderr, "window_update:  start\n");
01174 #endif
01175   gtk_widget_set_sensitive
01176     (GTK_WIDGET (window->button_evaluator),
01177      gtk_check_button_get_active (window->check_evaluator));
01178   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01179   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01180   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01181   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01182   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01183   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01184   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01185   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01186   gtk_widget_hide (GTK_WIDGET (window->label_population));
01187   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01188   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01189   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01190   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01191   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01192   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01193   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01194   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01195   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01196   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01197   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01198   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01199   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01200   gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01201   gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01202   gtk_widget_hide (GTK_WIDGET (window->label_step));
01203   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01204   gtk_widget_hide (GTK_WIDGET (window->label_p));
01205   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01206   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01207   switch (window_get_algorithm ())
01208     {
01209     case ALGORITHM_MONTE_CARLO:
01210       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01211       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01212       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01213       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01214       if (i > 1)
01215         {
01216           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01217           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01218           gtk_widget_show (GTK_WIDGET (window->label_bests));
01219           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01220         }
01221       window_update_climbing ();
01222       break;
01223     case ALGORITHM_SWEEP:
01224     case ALGORITHM_ORTHOGONAL:
01225       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01226       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01227       if (i > 1)
01228         {
01229           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01230           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01231           gtk_widget_show (GTK_WIDGET (window->label_bests));
01232           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01233         }
01234       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01235       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01236       gtk_widget_show (GTK_WIDGET (window->check_climbing));
01237       window_update_climbing ();
01238       break;
01239     default:
01240       gtk_widget_show (GTK_WIDGET (window->label_population));
01241       gtk_widget_show (GTK_WIDGET (window->spin_population));
01242       gtk_widget_show (GTK_WIDGET (window->label_generations));
01243       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01244       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01245       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
```

```
01246       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01247       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01248       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01249       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01250       gtk_widget_show (GTK_WIDGET (window->label_bits));
01251       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01252     }
01253   gtk_widget_set_sensitive
01254     (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
01255   gtk_widget_set_sensitive
01256     (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
01257   for (i = 0; i < input->experiment->ninputs; ++i)
01258     {
01259       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01260       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01261       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01262       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01263       g_signal_handler_block
01264         (window->check_template[i], window->id_template[i]);
01265       g_signal_handler_block (window->button_template[i], window->id_input[i]);
01266       gtk_check_button_set_active (window->check_template[i], 1);
01267       g_signal_handler_unblock (window->button_template[i],
01268                                 window->id_input[i]);
01269       g_signal_handler_unblock (window->check_template[i],
01270                                 window->id_template[i]);
01271     }
01272   if (i > 0)
01273     {
01274       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01275       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01276                                 gtk_check_button_get_active
01277                                 (window->check_template[i - 1]));
01278     }
01279   if (i < MAX_NINPUTS)
01280     {
01281       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01282       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01283       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01284       gtk_widget_set_sensitive
01285         (GTK_WIDGET (window->button_template[i]),
01286          gtk_check_button_get_active (window->check_template[i]));
01287       g_signal_handler_block
01288         (window->check_template[i], window->id_template[i]);
01289       g_signal_handler_block (window->button_template[i], window->id_input[i]);
01290       gtk_check_button_set_active (window->check_template[i], 0);
01291       g_signal_handler_unblock (window->button_template[i],
01292                                 window->id_input[i]);
01293       g_signal_handler_unblock (window->check_template[i],
01294                                 window->id_template[i]);
01295     }
01296   while (++i < MAX_NINPUTS)
01297     {
01298       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01299       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01300     }
01301   gtk_widget_set_sensitive
01302     (GTK_WIDGET (window->spin_minabs),
01303      gtk_check_button_get_active (window->check_minabs));
01304   gtk_widget_set_sensitive
01305     (GTK_WIDGET (window->spin_maxabs),
01306      gtk_check_button_get_active (window->check_maxabs));
01307   if (window_get_norm () == ERROR_NORM_P)
01308     {
01309       gtk_widget_show (GTK_WIDGET (window->label_p));
01310       gtk_widget_show (GTK_WIDGET (window->spin_p));
01311     }
01312 #if DEBUG_INTERFACE
01313   fprintf (stderr, "window_update:  end\n");
01314 #endif
01315 }
```

Here is the call graph for this function:



4.11.3.46 window_update_climbing()

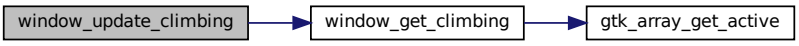
```
static void window_update_climbing ( )  [static]
```

Function to update hill climbing method widgets view in the main window.

Definition at line 1138 of file interface.c.

```
01139 {
01140 #if DEBUG_INTERFACE
01141   fprintf (stderr, "window_update_climbing:  start\n");
01142 #endif
01143   gtk_widget_show (GTK_WIDGET (window->check_climbing));
01144   if (gtk_check_button_get_active (window->check_climbing))
01145     {
01146       gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01147       gtk_widget_show (GTK_WIDGET (window->label_step));
01148       gtk_widget_show (GTK_WIDGET (window->spin_step));
01149     }
01150   switch (window_get_climbing ())
01151     {
01152     case CLIMBING_METHOD_COORDINATES:
01153       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01154       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01155       break;
01156     default:
01157       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01158       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01159     }
01160 #if DEBUG_INTERFACE
01161   fprintf (stderr, "window_update_climbing:  end\n");
01162 #endif
01163 }
```

Here is the call graph for this function:



4.11.3.47 window_update_variable()

```
static void window_update_variable ( )  [static]
```

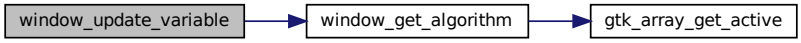
Function to update the variable data in the main window.

Definition at line 1921 of file interface.c.

```
01922 {
01923   int i;
01924 #if DEBUG_INTERFACE
01925   fprintf (stderr, "window_update_variable:  start\n");
01926 #endif
01927   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01928   if (i < 0)
01929     i = 0;
01930   switch (window_get_algorithm ())
01931     {
01932     case ALGORITHM_SWEEP:
01933     case ALGORITHM_ORTHOGONAL:
01934       input->variable[i].nsweeps
01935         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
```

```
01936 #if DEBUG_INTERFACE
01937       fprintf (stderr, "window_update_variable:  nsweeps[%d]=%u\n", i,
01938                input->variable[i].nsweeps);
01939 #endif
01940       break;
01941     case ALGORITHM_GENETIC:
01942       input->variable[i].nbits
01943         = gtk_spin_button_get_value_as_int (window->spin_bits);
01944 #if DEBUG_INTERFACE
01945       fprintf (stderr, "window_update_variable:  nbits[%d]=%u\n", i,
01946                input->variable[i].nbits);
01947 #endif
01948     }
01949 #if DEBUG_INTERFACE
01950   fprintf (stderr, "window_update_variable:  end\n");
01951 #endif
01952 }
```

Here is the call graph for this function:



4.11.3.48 window_weight_experiment()

```
static void window_weight_experiment ( )  [static]
```

Function to update the experiment weight in the main window.

Definition at line 1533 of file interface.c.

```
01534 {
01535   unsigned int i;
01536 #if DEBUG_INTERFACE
01537   fprintf (stderr, "window_weight_experiment:  start\n");
01538 #endif
01539   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01540   input->experiment[i].weight = gtk_spin_button_get_value (window->spin_weight);
01541 #if DEBUG_INTERFACE
01542   fprintf (stderr, "window_weight_experiment:  end\n");
01543 #endif
01544 }
```

4.11.4 Variable Documentation

4.11.4.1 logo

```
const char* logo[]  [static]
```

Logo pixmap.

Definition at line 84 of file interface.c.

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
00038 #define _GNU_SOURCE
```

```
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #include <json-glib/json-glib.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #include <gio/gio.h>
00057 #include <gtk/gtk.h>
00058 #include "jb/src/jb_xml.h"
00059 #include "jb/src/jb_json.h"
00060 #include "jb/src/jb_win.h"
00061 #include "genetic/genetic.h"
00062 #include "tools.h"
00063 #include "experiment.h"
00064 #include "variable.h"
00065 #include "input.h"
00066 #include "optimize.h"
00067 #include "interface.h"
00068
00069 #define DEBUG_INTERFACE 1
00070
00075 #ifdef G_OS_WIN32
00076 #define INPUT_FILE "test-ga-win.xml"
00077 #else
00078 #define INPUT_FILE "test-ga.xml"
00079 #endif
00080
00081 Window window[1];
00083
00084 static const char *logo[] = {
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 };
00122
00123 /*
00124 const char * logo[] = {
00125 "32 32 3 1",
00126 "     c #FFFFFFFFFFFF",
00127 ".        c #00000000FFFF",
00128 "X    c #FFFF00000000",
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
00237 static inline void
00238 input_save_xml (xmlDoc * doc)
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->seed != DEFAULT_RANDOM_SEED)
00279     jb_xml_node_set_uint (node, (const xmlChar *) LABEL_SEED, input->seed);
00280
00281   // Setting the algorithm
00282   buffer = (char *) g_slice_alloc (64);
00283   switch (input->algorithm)
00284     {
00285     case ALGORITHM_MONTE_CARLO:
00286       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00287                   (const xmlChar *) LABEL_MONTE_CARLO);
00288       snprintf (buffer, 64, "%u", input->nsimulations);
00289       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00290                   (xmlChar *) buffer);
00291       snprintf (buffer, 64, "%u", input->niterations);
00292       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00293                   (xmlChar *) buffer);
00294       snprintf (buffer, 64, "%.3lg", input->tolerance);
00295       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00296       snprintf (buffer, 64, "%u", input->nbest);
00297       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00298       input_save_climbing_xml (node);
00299       break;
00300     case ALGORITHM_SWEEP:
00301       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00302                   (const xmlChar *) LABEL_SWEEP);
00303       snprintf (buffer, 64, "%u", input->niterations);
00304       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00305                   (xmlChar *) buffer);
00306       snprintf (buffer, 64, "%.3lg", input->tolerance);
00307       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00308       snprintf (buffer, 64, "%u", input->nbest);
00309       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00310       input_save_climbing_xml (node);
00311       break;
00312     case ALGORITHM_ORTHOGONAL:
00313       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00314                   (const xmlChar *) LABEL_ORTHOGONAL);
00315       snprintf (buffer, 64, "%u", input->niterations);
```

```
00316       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00317                   (xmlChar *) buffer);
00318       snprintf (buffer, 64, "%.3lg", input->tolerance);
00319       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00320       snprintf (buffer, 64, "%u", input->nbest);
00321       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00322       input_save_climbing_xml (node);
00323       break;
00324     default:
00325       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00326                   (const xmlChar *) LABEL_GENETIC);
00327       snprintf (buffer, 64, "%u", input->nsimulations);
00328       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00329                   (xmlChar *) buffer);
00330       snprintf (buffer, 64, "%u", input->niterations);
00331       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00332                   (xmlChar *) buffer);
00333       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00334       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00335       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00336       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00337                   (xmlChar *) buffer);
00338       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00339       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00340       break;
00341     }
00342   g_slice_free1 (64, buffer);
00343   if (input->threshold != 0.)
00344     jb_xml_node_set_float (node, (const xmlChar *) LABEL_THRESHOLD,
00345                            input->threshold);
00346
00347   // Setting the experimental data
00348   for (i = 0; i < input->nexperiments; ++i)
00349     {
00350       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00351       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00352                   (xmlChar *) input->experiment[i].name);
00353       if (input->experiment[i].weight != 1.)
00354         jb_xml_node_set_float (child, (const xmlChar *) LABEL_WEIGHT,
00355                                input->experiment[i].weight);
00356       for (j = 0; j < input->experiment->ninputs; ++j)
00357         xmlSetProp (child, (const xmlChar *) stencil[j],
00358                     (xmlChar *) input->experiment[i].stencil[j]);
00359     }
00360
00361   // Setting the variables data
00362   for (i = 0; i < input->nvariables; ++i)
00363     {
00364       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00365       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00366                   (xmlChar *) input->variable[i].name);
00367       jb_xml_node_set_float (child, (const xmlChar *) LABEL_MINIMUM,
00368                              input->variable[i].rangemin);
00369       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00370         jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM,
00371                                input->variable[i].rangeminabs);
00372       jb_xml_node_set_float (child, (const xmlChar *) LABEL_MAXIMUM,
00373                              input->variable[i].rangemax);
00374       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00375         jb_xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM,
00376                                input->variable[i].rangemaxabs);
00377       if (input->variable[i].precision != DEFAULT_PRECISION)
00378         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_PRECISION,
00379                               input->variable[i].precision);
00380       if (input->algorithm == ALGORITHM_SWEEP
00381           || input->algorithm == ALGORITHM_ORTHOGONAL)
00382         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NSWEEPS,
00383                               input->variable[i].nsweeps);
00384       else if (input->algorithm == ALGORITHM_GENETIC)
00385         jb_xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00386                               input->variable[i].nbits);
00387       if (input->nsteps)
00388         jb_xml_node_set_float (child, (const xmlChar *) LABEL_STEP,
00389                                input->variable[i].step);
00390     }
00391
00392   // Saving the error norm
00393   switch (input->norm)
00394     {
00395     case ERROR_NORM_MAXIMUM:
00396       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397                   (const xmlChar *) LABEL_MAXIMUM);
00398       break;
00399     case ERROR_NORM_P:
00400       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                   (const xmlChar *) LABEL_P);
00402       jb_xml_node_set_float (node, (const xmlChar *) LABEL_P, input->p);
```

```
00403       break;
00404     case ERROR_NORM_TAXICAB:
00405       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00406                   (const xmlChar *) LABEL_TAXICAB);
00407     }
00408
00409 #if DEBUG_INTERFACE
00410   fprintf (stderr, "input_save:  end\n");
00411 #endif
00412 }
00413
00417 static inline void
00418 input_save_json (JsonGenerator * generator)
00419 {
00420   unsigned int i, j;
00421   char *buffer;
00422   JsonNode *node, *child;
00423   JsonObject *object;
00424   JsonArray *array;
00425   GFile *file, *file2;
00426
00427 #if DEBUG_INTERFACE
00428   fprintf (stderr, "input_save_json:  start\n");
00429 #endif
00430
00431   // Setting root JSON node
00432   object = json_object_new ();
00433   node = json_node_new (JSON_NODE_OBJECT);
00434   json_node_set_object (node, object);
00435   json_generator_set_root (generator, node);
00436
00437   // Adding properties to the root JSON node
00438   if (strcmp (input->result, result_name))
00439     json_object_set_string_member (object, LABEL_RESULT_FILE, input->result);
00440   if (strcmp (input->variables, variables_name))
00441     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00442                                    input->variables);
00443   file = g_file_new_for_path (input->directory);
00444   file2 = g_file_new_for_path (input->simulator);
00445   buffer = g_file_get_relative_path (file, file2);
00446   g_object_unref (file2);
00447   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00448   g_free (buffer);
00449   if (input->evaluator)
00450     {
00451       file2 = g_file_new_for_path (input->evaluator);
00452       buffer = g_file_get_relative_path (file, file2);
00453       g_object_unref (file2);
00454       if (strlen (buffer))
00455         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00456       g_free (buffer);
00457     }
00458   if (input->seed != DEFAULT_RANDOM_SEED)
00459     jb_json_object_set_uint (object, LABEL_SEED, input->seed);
00460
00461   // Setting the algorithm
00462   buffer = (char *) g_slice_alloc (64);
00463   switch (input->algorithm)
00464     {
00465     case ALGORITHM_MONTE_CARLO:
00466       json_object_set_string_member (object, LABEL_ALGORITHM,
00467                                      LABEL_MONTE_CARLO);
00468       snprintf (buffer, 64, "%u", input->nsimulations);
00469       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00470       snprintf (buffer, 64, "%u", input->niterations);
00471       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00472       snprintf (buffer, 64, "%.3lg", input->tolerance);
00473       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00474       snprintf (buffer, 64, "%u", input->nbest);
00475       json_object_set_string_member (object, LABEL_NBEST, buffer);
00476       input_save_climbing_json (node);
00477       break;
00478     case ALGORITHM_SWEEP:
00479       json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_SWEEP);
00480       snprintf (buffer, 64, "%u", input->niterations);
00481       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00482       snprintf (buffer, 64, "%.3lg", input->tolerance);
00483       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00484       snprintf (buffer, 64, "%u", input->nbest);
00485       json_object_set_string_member (object, LABEL_NBEST, buffer);
00486       input_save_climbing_json (node);
00487       break;
00488     case ALGORITHM_ORTHOGONAL:
00489       json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_ORTHOGONAL);
00490       snprintf (buffer, 64, "%u", input->niterations);
00491       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00492       snprintf (buffer, 64, "%.3lg", input->tolerance);
```

```
00493       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00494       snprintf (buffer, 64, "%u", input->nbest);
00495       json_object_set_string_member (object, LABEL_NBEST, buffer);
00496       input_save_climbing_json (node);
00497       break;
00498     default:
00499       json_object_set_string_member (object, LABEL_ALGORITHM, LABEL_GENETIC);
00500       snprintf (buffer, 64, "%u", input->nsimulations);
00501       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00502       snprintf (buffer, 64, "%u", input->niterations);
00503       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00504       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00505       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00506       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00507       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00508       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00509       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00510       break;
00511     }
00512   g_slice_free1 (64, buffer);
00513   if (input->threshold != 0.)
00514     jb_json_object_set_float (object, LABEL_THRESHOLD, input->threshold);
00515
00516   // Setting the experimental data
00517   array = json_array_new ();
00518   for (i = 0; i < input->nexperiments; ++i)
00519     {
00520       child = json_node_new (JSON_NODE_OBJECT);
00521       object = json_node_get_object (child);
00522       json_object_set_string_member (object, LABEL_NAME,
00523                                      input->experiment[i].name);
00524       if (input->experiment[i].weight != 1.)
00525         jb_json_object_set_float (object, LABEL_WEIGHT,
00526                                   input->experiment[i].weight);
00527       for (j = 0; j < input->experiment->ninputs; ++j)
00528         json_object_set_string_member (object, stencil[j],
00529                                        input->experiment[i].stencil[j]);
00530       json_array_add_element (array, child);
00531     }
00532   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00533
00534   // Setting the variables data
00535   array = json_array_new ();
00536   for (i = 0; i < input->nvariables; ++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->variable[i].name);
00542       jb_json_object_set_float (object, LABEL_MINIMUM,
00543                                 input->variable[i].rangemin);
00544       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00545         jb_json_object_set_float (object, LABEL_ABSOLUTE_MINIMUM,
00546                                   input->variable[i].rangeminabs);
00547       jb_json_object_set_float (object, LABEL_MAXIMUM,
00548                                 input->variable[i].rangemax);
00549       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00550         jb_json_object_set_float (object, LABEL_ABSOLUTE_MAXIMUM,
00551                                   input->variable[i].rangemaxabs);
00552       if (input->variable[i].precision != DEFAULT_PRECISION)
00553         jb_json_object_set_uint (object, LABEL_PRECISION,
00554                                  input->variable[i].precision);
00555       if (input->algorithm == ALGORITHM_SWEEP
00556           || input->algorithm == ALGORITHM_ORTHOGONAL)
00557         jb_json_object_set_uint (object, LABEL_NSWEEPS,
00558                                  input->variable[i].nsweeps);
00559       else if (input->algorithm == ALGORITHM_GENETIC)
00560         jb_json_object_set_uint (object, LABEL_NBITS, input->variable[i].nbits);
00561       if (input->nsteps)
00562         jb_json_object_set_float (object, LABEL_STEP, input->variable[i].step);
00563       json_array_add_element (array, child);
00564     }
00565   json_object_set_array_member (object, LABEL_VARIABLES, array);
00566
00567   // Saving the error norm
00568   switch (input->norm)
00569     {
00570     case ERROR_NORM_MAXIMUM:
00571       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00572       break;
00573     case ERROR_NORM_P:
00574       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00575       jb_json_object_set_float (object, LABEL_P, input->p);
00576       break;
00577     case ERROR_NORM_TAXICAB:
00578       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00579     }
```

```
00580
00581 #if DEBUG_INTERFACE
00582   fprintf (stderr, "input_save_json:  end\n");
00583 #endif
00584 }
00585
00589 static inline void
00590 input_save (char *filename)
00591 {
00592   xmlDoc *doc;
00593   JsonGenerator *generator;
00594
00595 #if DEBUG_INTERFACE
00596   fprintf (stderr, "input_save:  start\n");
00597 #endif
00598
00599   // Getting the input file directory
00600   input->name = g_path_get_basename (filename);
00601   input->directory = g_path_get_dirname (filename);
00602
00603   if (input->type == INPUT_TYPE_XML)
00604     {
00605       // Opening the input file
00606       doc = xmlNewDoc ((const xmlChar *) "1.0");
00607       input_save_xml (doc);
00608
00609       // Saving the XML file
00610       xmlSaveFormatFile (filename, doc, 1);
00611
00612       // Freeing memory
00613       xmlFreeDoc (doc);
00614     }
00615   else
00616     {
00617       // Opening the input file
00618       generator = json_generator_new ();
00619       json_generator_set_pretty (generator, TRUE);
00620       input_save_json (generator);
00621
00622       // Saving the JSON file
00623       json_generator_to_file (generator, filename, NULL);
00624
00625       // Freeing memory
00626       g_object_unref (generator);
00627     }
00628
00629 #if DEBUG_INTERFACE
00630   fprintf (stderr, "input_save:  end\n");
00631 #endif
00632 }
00633
00637 static void
00638 dialog_options_close (GtkDialog * dlg,
00639                       int response_id)
00640 {
00641 #if DEBUG_INTERFACE
00642   fprintf (stderr, "dialog_options_close:  start\n");
00643 #endif
00644   if (response_id == GTK_RESPONSE_OK)
00645     {
00646       input->seed
00647         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00648       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00649       nthreads_climbing
00650         = gtk_spin_button_get_value_as_int (options->spin_climbing);
00651     }
00652   gtk_window_destroy (GTK_WINDOW (dlg));
00653 #if DEBUG_INTERFACE
00654   fprintf (stderr, "dialog_options_close:  end\n");
00655 #endif
00656 }
00657
00661 static void
00662 options_new ()
00663 {
00664 #if DEBUG_INTERFACE
00665   fprintf (stderr, "options_new:  start\n");
00666 #endif
00667   options->label_seed = (GtkLabel *)
00668     gtk_label_new (_("Pseudo-random numbers generator seed"));
00669   options->spin_seed = (GtkSpinButton *)
00670     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00671   gtk_widget_set_tooltip_text
00672     (GTK_WIDGET (options->spin_seed),
00673      _("Seed to init the pseudo-random numbers generator"));
00674   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
00675   options->label_threads = (GtkLabel *)
```

```
00676     gtk_label_new (_("Threads number for the stochastic algorithm"));
00677   options->spin_threads
00678     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00679   gtk_widget_set_tooltip_text
00680     (GTK_WIDGET (options->spin_threads),
00681      _("Number of threads to perform the calibration/optimization for "
00682        "the stochastic algorithm"));
00683   gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00684   options->label_climbing = (GtkLabel *)
00685     gtk_label_new (_("Threads number for the hill climbing method"));
00686   options->spin_climbing =
00687     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00688   gtk_widget_set_tooltip_text
00689     (GTK_WIDGET (options->spin_climbing),
00690      _("Number of threads to perform the calibration/optimization for the "
00691        "hill climbing method"));
00692   gtk_spin_button_set_value (options->spin_climbing,
00693                              (gdouble) nthreads_climbing);
00694   options->grid = (GtkGrid *) gtk_grid_new ();
00695   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00696   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00697   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00698                    0, 1, 1, 1);
00699   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00700                    1, 1, 1, 1);
00701   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
00702                    1);
00703   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
00704                    1);
00705 #if !GTK4
00706   gtk_widget_show_all (GTK_WIDGET (options->grid));
00707 #else
00708   gtk_widget_show (GTK_WIDGET (options->grid));
00709 #endif
00710   options->dialog = (GtkDialog *)
00711     gtk_dialog_new_with_buttons (_("Options"),
00712                                  window->window,
00713                                  GTK_DIALOG_MODAL,
00714                                  _("_OK"), GTK_RESPONSE_OK,
00715                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00716   gtk_box_append (GTK_BOX (gtk_dialog_get_content_area (options->dialog)),
00717                   GTK_WIDGET (options->grid));
00718   g_signal_connect (options->dialog, "response",
00719                     G_CALLBACK (dialog_options_close), NULL);
00720   gtk_window_present (GTK_WINDOW (options->dialog));
00721 #if DEBUG_INTERFACE
00722   fprintf (stderr, "options_new:  end\n");
00723 #endif
00724 }
00725
00729 static inline void
00730 running_new ()
00731 {
00732 #if DEBUG_INTERFACE
00733   fprintf (stderr, "running_new:  start\n");
00734 #endif
00735   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00736   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00737   running->grid = (GtkGrid *) gtk_grid_new ();
00738   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00739   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00740   running->dialog = (GtkDialog *)
00741     gtk_dialog_new_with_buttons (_("Calculating"),
00742                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00743   gtk_window_set_child (GTK_WINDOW
00744                         (gtk_dialog_get_content_area (running->dialog)),
00745                         GTK_WIDGET (running->grid));
00746   gtk_spinner_start (running->spinner);
00747 #if !GTK4
00748   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00749 #else
00750   gtk_widget_show (GTK_WIDGET (running->dialog));
00751 #endif
00752 #if DEBUG_INTERFACE
00753   fprintf (stderr, "running_new:  end\n");
00754 #endif
00755 }
00756
00762 static unsigned int
00763 window_get_algorithm ()
00764 {
00765   unsigned int i;
00766 #if DEBUG_INTERFACE
00767   fprintf (stderr, "window_get_algorithm:  start\n");
00768 #endif
00769   i = gtk_array_get_active (window->button_algorithm, NALGORITHMS);
00770 #if DEBUG_INTERFACE
```

```
00771   fprintf (stderr, "window_get_algorithm:  %u\n", i);
00772   fprintf (stderr, "window_get_algorithm:  end\n");
00773 #endif
00774   return i;
00775 }
00776
00782 static unsigned int
00783 window_get_climbing ()
00784 {
00785   unsigned int i;
00786 #if DEBUG_INTERFACE
00787   fprintf (stderr, "window_get_climbing:  start\n");
00788 #endif
00789   i = gtk_array_get_active (window->button_climbing, NCLIMBINGS);
00790 #if DEBUG_INTERFACE
00791   fprintf (stderr, "window_get_climbing:  %u\n", i);
00792   fprintf (stderr, "window_get_climbing:  end\n");
00793 #endif
00794   return i;
00795 }
00796
00802 static unsigned int
00803 window_get_norm ()
00804 {
00805   unsigned int i;
00806 #if DEBUG_INTERFACE
00807   fprintf (stderr, "window_get_norm:  start\n");
00808 #endif
00809   i = gtk_array_get_active (window->button_norm, NNORMS);
00810 #if DEBUG_INTERFACE
00811   fprintf (stderr, "window_get_norm:  %u\n", i);
00812   fprintf (stderr, "window_get_norm:  end\n");
00813 #endif
00814   return i;
00815 }
00816
00820 static void
00821 window_save_climbing ()
00822 {
00823 #if DEBUG_INTERFACE
00824   fprintf (stderr, "window_save_climbing:  start\n");
00825 #endif
00826   if (gtk_check_button_get_active (window->check_climbing))
00827     {
00828       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
00829       input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
00830       switch (window_get_climbing ())
00831         {
00832         case CLIMBING_METHOD_COORDINATES:
00833           input->climbing = CLIMBING_METHOD_COORDINATES;
00834           break;
00835         default:
00836           input->climbing = CLIMBING_METHOD_RANDOM;
00837           input->nestimates
00838             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00839         }
00840     }
00841   else
00842     input->nsteps = 0;
00843 #if DEBUG_INTERFACE
00844   fprintf (stderr, "window_save_climbing:  end\n");
00845 #endif
00846 }
00847
00851 static void
00852 dialog_save_close (GtkFileChooserDialog * dlg,
00854                    int response_id)
00855 {
00856   GtkFileFilter *filter1;
00857   char *buffer;
00858 #if DEBUG_INTERFACE
00859   fprintf (stderr, "dialog_save_close:  start\n");
00860 #endif
00861   // If OK response then saving
00862   if (response_id == GTK_RESPONSE_OK)
00863     {
00864       // Setting input file type
00865       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866       buffer = (char *) gtk_file_filter_get_name (filter1);
00867       if (!strcmp (buffer, "XML"))
00868         input->type = INPUT_TYPE_XML;
00869       else
00870         input->type = INPUT_TYPE_JSON;
00871
00872       // Adding properties to the root XML node
00873       input->simulator
00874         = g_strdup (gtk_button_get_label (window->button_simulator));
```

```
00875       if (gtk_check_button_get_active (window->check_evaluator))
00876         input->evaluator
00877           = g_strdup (gtk_button_get_label (window->button_evaluator));
00878       else
00879         input->evaluator = NULL;
00880       if (input->type == INPUT_TYPE_XML)
00881         {
00882           input->result
00883             = (char *) xmlStrdup ((const xmlChar *)
00884                                   gtk_entry_get_text (window->entry_result));
00885           input->variables
00886             = (char *) xmlStrdup ((const xmlChar *)
00887                                   gtk_entry_get_text (window->entry_variables));
00888         }
00889       else
00890         {
00891           input->result = g_strdup (gtk_entry_get_text (window->entry_result));
00892           input->variables =
00893             g_strdup (gtk_entry_get_text (window->entry_variables));
00894         }
00895
00896       // Setting the algorithm
00897       switch (window_get_algorithm ())
00898         {
00899         case ALGORITHM_MONTE_CARLO:
00900           input->algorithm = ALGORITHM_MONTE_CARLO;
00901           input->nsimulations
00902             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00903           input->niterations
00904             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00905           input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00906           input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00907           window_save_climbing ();
00908           break;
00909         case ALGORITHM_SWEEP:
00910           input->algorithm = ALGORITHM_SWEEP;
00911           input->niterations
00912             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00913           input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00914           input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00915           window_save_climbing ();
00916           break;
00917         case ALGORITHM_ORTHOGONAL:
00918           input->algorithm = ALGORITHM_ORTHOGONAL;
00919           input->niterations
00920             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921           input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
00922           input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
00923           window_save_climbing ();
00924           break;
00925         default:
00926           input->algorithm = ALGORITHM_GENETIC;
00927           input->nsimulations
00928             = gtk_spin_button_get_value_as_int (window->spin_population);
00929           input->niterations
00930             = gtk_spin_button_get_value_as_int (window->spin_generations);
00931           input->mutation_ratio
00932             = gtk_spin_button_get_value (window->spin_mutation);
00933           input->reproduction_ratio
00934             = gtk_spin_button_get_value (window->spin_reproduction);
00935           input->adaptation_ratio
00936             = gtk_spin_button_get_value (window->spin_adaptation);
00937         }
00938       input->norm = window_get_norm ();
00939       input->p = gtk_spin_button_get_value (window->spin_p);
00940       input->threshold = gtk_spin_button_get_value (window->spin_threshold);
00941
00942       // Saving the XML file
00943       buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
00944       input_save (buffer);
00945
00946       // Closing and freeing memory
00947       g_free (buffer);
00948     }
00949   gtk_window_destroy (GTK_WINDOW (dlg));
00950 #if DEBUG_INTERFACE
00951   fprintf (stderr, "dialog_save_close:  end\n");
00952 #endif
00953 }
00954
00958 static void
00959 window_save ()
00960 {
00961   GtkFileChooserDialog *dlg;
00962   GtkFileFilter *filter1, *filter2;
00963   char *buffer;
00964
```

```
00965 #if DEBUG_INTERFACE
00966   fprintf (stderr, "window_save:  start\n");
00967 #endif
00968
00969   // Opening the saving dialog
00970   dlg = (GtkFileChooserDialog *)
00971     gtk_file_chooser_dialog_new (_("Save file"),
00972                                  window->window,
00973                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00974                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00975                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00976 #if !GTK4
00977   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00978 #endif
00979   buffer = g_build_filename (input->directory, input->name, NULL);
00980   gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
00981   g_free (buffer);
00982
00983   // Adding XML filter
00984   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00985   gtk_file_filter_set_name (filter1, "XML");
00986   gtk_file_filter_add_pattern (filter1, "*.xml");
00987   gtk_file_filter_add_pattern (filter1, "*.XML");
00988   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00989
00990   // Adding JSON filter
00991   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00992   gtk_file_filter_set_name (filter2, "JSON");
00993   gtk_file_filter_add_pattern (filter2, "*.json");
00994   gtk_file_filter_add_pattern (filter2, "*.JSON");
00995   gtk_file_filter_add_pattern (filter2, "*.js");
00996   gtk_file_filter_add_pattern (filter2, "*.JS");
00997   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00998
00999   if (input->type == INPUT_TYPE_XML)
01000     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
01001   else
01002     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
01003
01004   // Connecting the close function
01005   g_signal_connect (dlg, "response", G_CALLBACK (dialog_save_close), NULL);
01006
01007   // Showing modal dialog
01008   gtk_window_present (GTK_WINDOW (dlg));
01009
01010 #if DEBUG_INTERFACE
01011   fprintf (stderr, "window_save:  end\n");
01012 #endif
01013 }
01014
01018 static void
01019 window_run ()
01020 {
01021   GMainContext *context;
01022   char *msg, *msg2, buffer[64], buffer2[64];
01023   unsigned int i;
01024 #if DEBUG_INTERFACE
01025   fprintf (stderr, "window_run:  start\n");
01026 #endif
01027   window_save ();
01028   running_new ();
01029   context = g_main_context_default ();
01030   while (g_main_context_pending (context))
01031     g_main_context_iteration (context, 0);
01032   optimize_open ();
01033 #if DEBUG_INTERFACE
01034   fprintf (stderr, "window_run:  closing running dialog\n");
01035 #endif
01036   gtk_spinner_stop (running->spinner);
01037   gtk_window_destroy (GTK_WINDOW (running->dialog));
01038 #if DEBUG_INTERFACE
01039   fprintf (stderr, "window_run:  displaying results\n");
01040 #endif
01041   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01042   msg2 = g_strdup (buffer);
01043   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01044     {
01045       snprintf (buffer, 64, "%s = %s\n",
01046                 input->variable[i].name, format[input->variable[i].precision]);
01047       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01048       msg = g_strconcat (msg2, buffer2, NULL);
01049       g_free (msg2);
01050     }
01051   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01052             optimize->calculation_time);
01053   msg = g_strconcat (msg2, buffer, NULL);
01054   g_free (msg2);
```

```
01055   jbw_show_message (_("Best result"), msg, INFO_TYPE);
01056   g_free (msg);
01057 #if DEBUG_INTERFACE
01058   fprintf (stderr, "window_run:  freeing memory\n");
01059 #endif
01060   optimize_free ();
01061 #if DEBUG_INTERFACE
01062   fprintf (stderr, "window_run:  end\n");
01063 #endif
01064 }
01065
01069 static void
01070 window_help ()
01071 {
01072   char *buffer, *buffer2;
01073 #if DEBUG_INTERFACE
01074   fprintf (stderr, "window_help:  start\n");
01075 #endif
01076   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01077                               _("user-manual.pdf"), NULL);
01078   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01079   g_free (buffer2);
01080 #if !GTK4
01081 #if GTK_MINOR_VERSION >= 22
01082   gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01083 #else
01084   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01085 #endif
01086 #else
01087   gtk_show_uri (window->window, buffer, GDK_CURRENT_TIME);
01088 #endif
01089 #if DEBUG_INTERFACE
01090   fprintf (stderr, "window_help:  uri=%s\n", buffer);
01091 #endif
01092   g_free (buffer);
01093 #if DEBUG_INTERFACE
01094   fprintf (stderr, "window_help:  end\n");
01095 #endif
01096 }
01097
01101 static void
01102 window_about ()
01103 {
01104   static const gchar *authors[] = {
01105     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01106     "Borja Latorre Garcés <borja.latorre@csic.es>",
01107     NULL
01108   };
01109 #if DEBUG_INTERFACE
01110   fprintf (stderr, "window_about:  start\n");
01111 #endif
01112   gtk_show_about_dialog
01113     (window->window,
01114      "program_name", "MPCOTool",
01115      "comments",
01116      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01117        "A software to perform calibrations or optimizations of empirical "
01118        "parameters"),
01119      "authors", authors,
01120      "translator-credits",
01121      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01122      "(english, french and spanish)\n"
01123      "Uğur Çayoğlu (german)",
01124      "version", "4.4.1",
01125      "copyright", "Copyright 2012-2023 Javier Burguete Tolosa",
01126      "logo", window->logo,
01127      "website", "https://github.com/jburguete/mpcotool",
01128      "license-type", GTK_LICENSE_BSD, NULL);
01129 #if DEBUG_INTERFACE
01130   fprintf (stderr, "window_about:  end\n");
01131 #endif
01132 }
01133
01137 static void
01138 window_update_climbing ()
01139 {
01140 #if DEBUG_INTERFACE
01141   fprintf (stderr, "window_update_climbing:  start\n");
01142 #endif
01143   gtk_widget_show (GTK_WIDGET (window->check_climbing));
01144   if (gtk_check_button_get_active (window->check_climbing))
01145     {
01146       gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01147       gtk_widget_show (GTK_WIDGET (window->label_step));
01148       gtk_widget_show (GTK_WIDGET (window->spin_step));
01149     }
01150   switch (window_get_climbing ())
```

```
01151     {
01152     case CLIMBING_METHOD_COORDINATES:
01153       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01154       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01155       break;
01156     default:
01157       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01158       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01159     }
01160 #if DEBUG_INTERFACE
01161   fprintf (stderr, "window_update_climbing:  end\n");
01162 #endif
01163 }
01164
01168 static void
01169 window_update ()
01170 {
01171   unsigned int i;
01172 #if DEBUG_INTERFACE
01173   fprintf (stderr, "window_update:  start\n");
01174 #endif
01175   gtk_widget_set_sensitive
01176     (GTK_WIDGET (window->button_evaluator),
01177      gtk_check_button_get_active (window->check_evaluator));
01178   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01179   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01180   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01181   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01182   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01183   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01184   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01185   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01186   gtk_widget_hide (GTK_WIDGET (window->label_population));
01187   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01188   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01189   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01190   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01191   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01192   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01193   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01194   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01195   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01196   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01197   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01198   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01199   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01200   gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01201   gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01202   gtk_widget_hide (GTK_WIDGET (window->label_step));
01203   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01204   gtk_widget_hide (GTK_WIDGET (window->label_p));
01205   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01206   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01207   switch (window_get_algorithm ())
01208     {
01209     case ALGORITHM_MONTE_CARLO:
01210       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01211       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01212       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01213       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01214       if (i > 1)
01215         {
01216           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01217           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01218           gtk_widget_show (GTK_WIDGET (window->label_bests));
01219           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01220         }
01221       window_update_climbing ();
01222       break;
01223     case ALGORITHM_SWEEP:
01224     case ALGORITHM_ORTHOGONAL:
01225       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01226       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01227       if (i > 1)
01228         {
01229           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01230           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01231           gtk_widget_show (GTK_WIDGET (window->label_bests));
01232           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01233         }
01234       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01235       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01236       gtk_widget_show (GTK_WIDGET (window->check_climbing));
01237       window_update_climbing ();
01238       break;
01239     default:
01240       gtk_widget_show (GTK_WIDGET (window->label_population));
```

```
01241       gtk_widget_show (GTK_WIDGET (window->spin_population));
01242       gtk_widget_show (GTK_WIDGET (window->label_generations));
01243       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01244       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01245       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01246       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01247       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01248       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01249       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01250       gtk_widget_show (GTK_WIDGET (window->label_bits));
01251       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01252     }
01253   gtk_widget_set_sensitive
01254     (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
01255   gtk_widget_set_sensitive
01256     (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
01257   for (i = 0; i < input->experiment->ninputs; ++i)
01258     {
01259       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01260       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01261       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01262       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01263       g_signal_handler_block
01264         (window->check_template[i], window->id_template[i]);
01265       g_signal_handler_block (window->button_template[i], window->id_input[i]);
01266       gtk_check_button_set_active (window->check_template[i], 1);
01267       g_signal_handler_unblock (window->button_template[i],
01268                                 window->id_input[i]);
01269       g_signal_handler_unblock (window->check_template[i],
01270                                 window->id_template[i]);
01271     }
01272   if (i > 0)
01273     {
01274       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01275       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01276                                 gtk_check_button_get_active
01277                                 (window->check_template[i - 1]));
01278     }
01279   if (i < MAX_NINPUTS)
01280     {
01281       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01282       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01283       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01284       gtk_widget_set_sensitive
01285         (GTK_WIDGET (window->button_template[i]),
01286          gtk_check_button_get_active (window->check_template[i]));
01287       g_signal_handler_block
01288         (window->check_template[i], window->id_template[i]);
01289       g_signal_handler_block (window->button_template[i], window->id_input[i]);
01290       gtk_check_button_set_active (window->check_template[i], 0);
01291       g_signal_handler_unblock (window->button_template[i],
01292                                 window->id_input[i]);
01293       g_signal_handler_unblock (window->check_template[i],
01294                                 window->id_template[i]);
01295     }
01296   while (++i < MAX_NINPUTS)
01297     {
01298       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01299       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01300     }
01301   gtk_widget_set_sensitive
01302     (GTK_WIDGET (window->spin_minabs),
01303      gtk_check_button_get_active (window->check_minabs));
01304   gtk_widget_set_sensitive
01305     (GTK_WIDGET (window->spin_maxabs),
01306      gtk_check_button_get_active (window->check_maxabs));
01307   if (window_get_norm () == ERROR_NORM_P)
01308     {
01309       gtk_widget_show (GTK_WIDGET (window->label_p));
01310       gtk_widget_show (GTK_WIDGET (window->spin_p));
01311     }
01312 #if DEBUG_INTERFACE
01313   fprintf (stderr, "window_update:  end\n");
01314 #endif
01315 }
01316
01320 static void
01321 window_set_algorithm ()
01322 {
01323   int i;
01324 #if DEBUG_INTERFACE
01325   fprintf (stderr, "window_set_algorithm:  start\n");
01326 #endif
01327   i = window_get_algorithm ();
01328   switch (i)
01329     {
01330     case ALGORITHM_SWEEP:
```

```
01331     case ALGORITHM_ORTHOGONAL:
01332       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01333       if (i < 0)
01334         i = 0;
01335       gtk_spin_button_set_value (window->spin_sweeps,
01336                                  (gdouble) input->variable[i].nsweeps);
01337       break;
01338     case ALGORITHM_GENETIC:
01339       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01340       if (i < 0)
01341         i = 0;
01342       gtk_spin_button_set_value (window->spin_bits,
01343                                  (gdouble) input->variable[i].nbits);
01344     }
01345   window_update ();
01346 #if DEBUG_INTERFACE
01347   fprintf (stderr, "window_set_algorithm:  end\n");
01348 #endif
01349 }
01350
01354 static void
01355 window_set_experiment ()
01356 {
01357   unsigned int i, j;
01358   char *buffer1;
01359 #if DEBUG_INTERFACE
01360   fprintf (stderr, "window_set_experiment:  start\n");
01361 #endif
01362   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01363   gtk_spin_button_set_value (window->spin_weight, input->experiment[i].weight);
01364   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01365   gtk_button_set_label (window->button_experiment, buffer1);
01366   g_free (buffer1);
01367   for (j = 0; j < input->experiment->ninputs; ++j)
01368     {
01369       g_signal_handler_block (window->button_template[j], window->id_input[j]);
01370       gtk_button_set_label (window->button_template[j],
01371                             input->experiment[i].stencil[j]);
01372       g_signal_handler_unblock
01373         (window->button_template[j], window->id_input[j]);
01374     }
01375 #if DEBUG_INTERFACE
01376   fprintf (stderr, "window_set_experiment:  end\n");
01377 #endif
01378 }
01379
01383 static void
01384 window_remove_experiment ()
01385 {
01386   unsigned int i, j;
01387 #if DEBUG_INTERFACE
01388   fprintf (stderr, "window_remove_experiment:  start\n");
01389 #endif
01390   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01391   g_signal_handler_block (window->combo_experiment, window->id_experiment);
01392   gtk_combo_box_text_remove (window->combo_experiment, i);
01393   g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
01394   experiment_free (input->experiment + i, input->type);
01395   --input->nexperiments;
01396   for (j = i; j < input->nexperiments; ++j)
01397     memcpy (input->experiment + j, input->experiment + j + 1,
01398             sizeof (Experiment));
01399   j = input->nexperiments - 1;
01400   if (i > j)
01401     i = j;
01402   for (j = 0; j < input->experiment->ninputs; ++j)
01403     g_signal_handler_block (window->button_template[j], window->id_input[j]);
01404   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01405   for (j = 0; j < input->experiment->ninputs; ++j)
01406     g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
01407   window_update ();
01408 #if DEBUG_INTERFACE
01409   fprintf (stderr, "window_remove_experiment:  end\n");
01410 #endif
01411 }
01412
01416 static void
01417 window_add_experiment ()
01418 {
01419   unsigned int i, j;
01420 #if DEBUG_INTERFACE
01421   fprintf (stderr, "window_add_experiment:  start\n");
01422 #endif
01423   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01424   g_signal_handler_block (window->combo_experiment, window->id_experiment);
01425   gtk_combo_box_text_insert_text
01426     (window->combo_experiment, i, input->experiment[i].name);
```

```
01427   g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
01428   input->experiment = (Experiment *) g_realloc
01429     (input->experiment, (input->nexperiments + 1) * sizeof (Experiment));
01430   for (j = input->nexperiments - 1; j > i; --j)
01431     memcpy (input->experiment + j + 1, input->experiment + j,
01432             sizeof (Experiment));
01433   input->experiment[j + 1].weight = input->experiment[j].weight;
01434   input->experiment[j + 1].ninputs = input->experiment[j].ninputs;
01435   if (input->type == INPUT_TYPE_XML)
01436     {
01437       input->experiment[j + 1].name
01438         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
01439       for (j = 0; j < input->experiment->ninputs; ++j)
01440         input->experiment[i + 1].stencil[j]
01441           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].stencil[j]);
01442     }
01443   else
01444     {
01445       input->experiment[j + 1].name = g_strdup (input->experiment[j].name);
01446       for (j = 0; j < input->experiment->ninputs; ++j)
01447         input->experiment[i + 1].stencil[j]
01448           = g_strdup (input->experiment[i].stencil[j]);
01449     }
01450   ++input->nexperiments;
01451   for (j = 0; j < input->experiment->ninputs; ++j)
01452     g_signal_handler_block (window->button_template[j], window->id_input[j]);
01453   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01454   for (j = 0; j < input->experiment->ninputs; ++j)
01455     g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
01456   window_update ();
01457 #if DEBUG_INTERFACE
01458   fprintf (stderr, "window_add_experiment:  end\n");
01459 #endif
01460 }
01461
01465 static void
01466 dialog_name_experiment_close (GtkFileChooserDialog * dlg,
01468                               int response_id,
01469                               void *data)
01470 {
01471   char *buffer;
01472   unsigned int i;
01473 #if DEBUG_INTERFACE
01474   fprintf (stderr, "window_name_experiment_close:  start\n");
01475 #endif
01476   i = (size_t) data;
01477   if (response_id == GTK_RESPONSE_OK)
01478     {
01479       buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01480       g_signal_handler_block (window->combo_experiment, window->id_experiment);
01481       gtk_combo_box_text_remove (window->combo_experiment, i);
01482       gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01483       gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01484       g_signal_handler_unblock (window->combo_experiment,
01485                                 window->id_experiment);
01486       g_free (buffer);
01487     }
01488 #if DEBUG_INTERFACE
01489   fprintf (stderr, "window_name_experiment_close:  end\n");
01490 #endif
01491 }
01492
01496 static void
01497 window_name_experiment ()
01498 {
01499   GtkFileChooserDialog *dlg;
01500   GMainLoop *loop;
01501   const char *buffer;
01502   unsigned int i;
01503 #if DEBUG_INTERFACE
01504   fprintf (stderr, "window_name_experiment:  start\n");
01505 #endif
01506   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01507   buffer = gtk_button_get_label (window->button_experiment);
01508   dlg = (GtkFileChooserDialog *)
01509     gtk_file_chooser_dialog_new (_("Open experiment file"),
01510                                  window->window,
01511                                  GTK_FILE_CHOOSER_ACTION_OPEN,
01512                                  _("_Cancel"),
01513                                  GTK_RESPONSE_CANCEL,
01514                                  _("_Open"), GTK_RESPONSE_OK, NULL);
01515   gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01516   g_signal_connect (dlg, "response", G_CALLBACK (dialog_name_experiment_close),
01517                     (void *) (size_t) i);
01518   gtk_window_present (GTK_WINDOW (dlg));
01519   loop = g_main_loop_new (NULL, 0);
01520   g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
```

```
01521                               loop);
01522   g_main_loop_run (loop);
01523   g_main_loop_unref (loop);
01524 #if DEBUG_INTERFACE
01525   fprintf (stderr, "window_name_experiment:  end\n");
01526 #endif
01527 }
01528
01532 static void
01533 window_weight_experiment ()
01534 {
01535   unsigned int i;
01536 #if DEBUG_INTERFACE
01537   fprintf (stderr, "window_weight_experiment:  start\n");
01538 #endif
01539   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01540   input->experiment[i].weight = gtk_spin_button_get_value (window->spin_weight);
01541 #if DEBUG_INTERFACE
01542   fprintf (stderr, "window_weight_experiment:  end\n");
01543 #endif
01544 }
01545
01549 static void
01550 window_inputs_experiment ()
01551 {
01552   unsigned int j;
01553 #if DEBUG_INTERFACE
01554   fprintf (stderr, "window_inputs_experiment:  start\n");
01555 #endif
01556   j = input->experiment->ninputs - 1;
01557   if (j && !gtk_check_button_get_active (window->check_template[j]))
01558     --input->experiment->ninputs;
01559   if (input->experiment->ninputs < MAX_NINPUTS
01560       && gtk_check_button_get_active (window->check_template[j]))
01561     ++input->experiment->ninputs;
01562   window_update ();
01563 #if DEBUG_INTERFACE
01564   fprintf (stderr, "window_inputs_experiment:  end\n");
01565 #endif
01566 }
01567
01571 static void
01572 window_template_experiment_close (GtkFileChooserDialog * dlg,
01574                                   int response_id,
01576                                   void *data)
01577 {
01578   GFile *file1, *file2;
01579   char *buffer1, *buffer2;
01580   unsigned int i, j;
01581 #if DEBUG_INTERFACE
01582   fprintf (stderr, "window_template_experiment_close:  start\n");
01583 #endif
01584   if (response_id == GTK_RESPONSE_OK)
01585     {
01586       i = (size_t) data;
01587       j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01588       buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
01589       file1 = g_file_new_for_path (buffer1);
01590       file2 = g_file_new_for_path (input->directory);
01591       buffer2 = g_file_get_relative_path (file2, file1);
01592       if (input->type == INPUT_TYPE_XML)
01593         input->experiment[j].stencil[i]
01594           = (char *) xmlStrdup ((xmlChar *) buffer2);
01595       else
01596         input->experiment[j].stencil[i] = g_strdup (buffer2);
01597       g_free (buffer2);
01598       g_object_unref (file2);
01599       g_object_unref (file1);
01600       g_free (buffer1);
01601     }
01602   gtk_window_destroy (GTK_WINDOW (dlg));
01603 #if DEBUG_INTERFACE
01604   fprintf (stderr, "window_template_experiment_close:  end\n");
01605 #endif
01606 }
01607
01611 static void
01612 window_template_experiment (void *data)
01614 {
01615   GtkFileChooserDialog *dlg;
01616   GMainLoop *loop;
01617   const char *buffer;
01618   unsigned int i;
01619 #if DEBUG_INTERFACE
01620   fprintf (stderr, "window_template_experiment:  start\n");
01621 #endif
01622   i = (size_t) data;
```

```
01623   buffer = gtk_button_get_label (window->button_template[i]);
01624   dlg = (GtkFileChooserDialog *)
01625     gtk_file_chooser_dialog_new (_("Open template file"),
01626                                  window->window,
01627                                  GTK_FILE_CHOOSER_ACTION_OPEN,
01628                                  _("_Cancel"),
01629                                  GTK_RESPONSE_CANCEL,
01630                                  _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
01631   gtk_file_chooser_set_current_name (GTK_FILE_CHOOSER (dlg), buffer);
01632   g_signal_connect (dlg, "response",
01633                     G_CALLBACK (window_template_experiment_close), data);
01634   gtk_window_present (GTK_WINDOW (dlg));
01635   loop = g_main_loop_new (NULL, 0);
01636   g_signal_connect_swapped (dlg, "destroy", G_CALLBACK (g_main_loop_quit),
01637                             loop);
01638   g_main_loop_run (loop);
01639   g_main_loop_unref (loop);
01640 #if DEBUG_INTERFACE
01641   fprintf (stderr, "window_template_experiment:  end\n");
01642 #endif
01643 }
01644
01648 static void
01649 window_set_variable ()
01650 {
01651   unsigned int i;
01652 #if DEBUG_INTERFACE
01653   fprintf (stderr, "window_set_variable:  start\n");
01654 #endif
01655   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01656   g_signal_handler_block (window->entry_variable, window->id_variable_label);
01657   gtk_entry_set_text (window->entry_variable, input->variable[i].name);
01658   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01659   gtk_spin_button_set_value (window->spin_min, input->variable[i].rangemin);
01660   gtk_spin_button_set_value (window->spin_max, input->variable[i].rangemax);
01661   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01662     {
01663       gtk_spin_button_set_value (window->spin_minabs,
01664                                  input->variable[i].rangeminabs);
01665       gtk_check_button_set_active (window->check_minabs, 1);
01666     }
01667   else
01668     {
01669       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01670       gtk_check_button_set_active (window->check_minabs, 0);
01671     }
01672   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01673     {
01674       gtk_spin_button_set_value (window->spin_maxabs,
01675                                  input->variable[i].rangemaxabs);
01676       gtk_check_button_set_active (window->check_maxabs, 1);
01677     }
01678   else
01679     {
01680       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01681       gtk_check_button_set_active (window->check_maxabs, 0);
01682     }
01683   gtk_spin_button_set_value (window->spin_precision,
01684                              input->variable[i].precision);
01685   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
01686   if (input->nsteps)
01687     gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
01688 #if DEBUG_INTERFACE
01689   fprintf (stderr, "window_set_variable:  precision[%u]=%u\n", i,
01690            input->variable[i].precision);
01691 #endif
01692   switch (window_get_algorithm ())
01693     {
01694     case ALGORITHM_SWEEP:
01695     case ALGORITHM_ORTHOGONAL:
01696       gtk_spin_button_set_value (window->spin_sweeps,
01697                                  (gdouble) input->variable[i].nsweeps);
01698 #if DEBUG_INTERFACE
01699       fprintf (stderr, "window_set_variable:  nsweeps[%u]=%u\n", i,
01700                input->variable[i].nsweeps);
01701 #endif
01702       break;
01703     case ALGORITHM_GENETIC:
01704       gtk_spin_button_set_value (window->spin_bits,
01705                                  (gdouble) input->variable[i].nbits);
01706 #if DEBUG_INTERFACE
01707       fprintf (stderr, "window_set_variable:  nbits[%u]=%u\n", i,
01708                input->variable[i].nbits);
01709 #endif
01710       break;
01711     }
01712   window_update ();
```

```
01713 #if DEBUG_INTERFACE
01714   fprintf (stderr, "window_set_variable:  end\n");
01715 #endif
01716 }
01717
01721 static void
01722 window_remove_variable ()
01723 {
01724   unsigned int i, j;
01725 #if DEBUG_INTERFACE
01726   fprintf (stderr, "window_remove_variable:  start\n");
01727 #endif
01728   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01729   g_signal_handler_block (window->combo_variable, window->id_variable);
01730   gtk_combo_box_text_remove (window->combo_variable, i);
01731   g_signal_handler_unblock (window->combo_variable, window->id_variable);
01732   xmlFree (input->variable[i].name);
01733   --input->nvariables;
01734   for (j = i; j < input->nvariables; ++j)
01735     memcpy (input->variable + j, input->variable + j + 1, sizeof (Variable));
01736   j = input->nvariables - 1;
01737   if (i > j)
01738     i = j;
01739   g_signal_handler_block (window->entry_variable, window->id_variable_label);
01740   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01741   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01742   window_update ();
01743 #if DEBUG_INTERFACE
01744   fprintf (stderr, "window_remove_variable:  end\n");
01745 #endif
01746 }
01747
01751 static void
01752 window_add_variable ()
01753 {
01754   unsigned int i, j;
01755 #if DEBUG_INTERFACE
01756   fprintf (stderr, "window_add_variable:  start\n");
01757 #endif
01758   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01759   g_signal_handler_block (window->combo_variable, window->id_variable);
01760   gtk_combo_box_text_insert_text (window->combo_variable, i,
01761                                   input->variable[i].name);
01762   g_signal_handler_unblock (window->combo_variable, window->id_variable);
01763   input->variable = (Variable *) g_realloc
01764     (input->variable, (input->nvariables + 1) * sizeof (Variable));
01765   for (j = input->nvariables - 1; j > i; --j)
01766     memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01767   memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
01768   if (input->type == INPUT_TYPE_XML)
01769     input->variable[j + 1].name
01770       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01771   else
01772     input->variable[j + 1].name = g_strdup (input->variable[j].name);
01773   ++input->nvariables;
01774   g_signal_handler_block (window->entry_variable, window->id_variable_label);
01775   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01776   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
01777   window_update ();
01778 #if DEBUG_INTERFACE
01779   fprintf (stderr, "window_add_variable:  end\n");
01780 #endif
01781 }
01782
01786 static void
01787 window_label_variable ()
01788 {
01789   unsigned int i;
01790   const char *buffer;
01791 #if DEBUG_INTERFACE
01792   fprintf (stderr, "window_label_variable:  start\n");
01793 #endif
01794   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01795   buffer = gtk_entry_get_text (window->entry_variable);
01796   g_signal_handler_block (window->combo_variable, window->id_variable);
01797   gtk_combo_box_text_remove (window->combo_variable, i);
01798   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01799   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01800   g_signal_handler_unblock (window->combo_variable, window->id_variable);
01801 #if DEBUG_INTERFACE
01802   fprintf (stderr, "window_label_variable:  end\n");
01803 #endif
01804 }
01805
01809 static void
01810 window_precision_variable ()
01811 {
```

```
01812   unsigned int i;
01813 #if DEBUG_INTERFACE
01814   fprintf (stderr, "window_precision_variable:  start\n");
01815 #endif
01816   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01817   input->variable[i].precision
01818     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01819   gtk_spin_button_set_digits (window->spin_min, input->variable[i].precision);
01820   gtk_spin_button_set_digits (window->spin_max, input->variable[i].precision);
01821   gtk_spin_button_set_digits (window->spin_minabs,
01822                               input->variable[i].precision);
01823   gtk_spin_button_set_digits (window->spin_maxabs,
01824                               input->variable[i].precision);
01825 #if DEBUG_INTERFACE
01826   fprintf (stderr, "window_precision_variable:  end\n");
01827 #endif
01828 }
01829
01833 static void
01834 window_rangemin_variable ()
01835 {
01836   unsigned int i;
01837 #if DEBUG_INTERFACE
01838   fprintf (stderr, "window_rangemin_variable:  start\n");
01839 #endif
01840   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01841   input->variable[i].rangemin = gtk_spin_button_get_value (window->spin_min);
01842 #if DEBUG_INTERFACE
01843   fprintf (stderr, "window_rangemin_variable:  end\n");
01844 #endif
01845 }
01846
01850 static void
01851 window_rangemax_variable ()
01852 {
01853   unsigned int i;
01854 #if DEBUG_INTERFACE
01855   fprintf (stderr, "window_rangemax_variable:  start\n");
01856 #endif
01857   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01858   input->variable[i].rangemax = gtk_spin_button_get_value (window->spin_max);
01859 #if DEBUG_INTERFACE
01860   fprintf (stderr, "window_rangemax_variable:  end\n");
01861 #endif
01862 }
01863
01867 static void
01868 window_rangeminabs_variable ()
01869 {
01870   unsigned int i;
01871 #if DEBUG_INTERFACE
01872   fprintf (stderr, "window_rangeminabs_variable:  start\n");
01873 #endif
01874   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01875   input->variable[i].rangeminabs
01876     = gtk_spin_button_get_value (window->spin_minabs);
01877 #if DEBUG_INTERFACE
01878   fprintf (stderr, "window_rangeminabs_variable:  end\n");
01879 #endif
01880 }
01881
01885 static void
01886 window_rangemaxabs_variable ()
01887 {
01888   unsigned int i;
01889 #if DEBUG_INTERFACE
01890   fprintf (stderr, "window_rangemaxabs_variable:  start\n");
01891 #endif
01892   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01893   input->variable[i].rangemaxabs
01894     = gtk_spin_button_get_value (window->spin_maxabs);
01895 #if DEBUG_INTERFACE
01896   fprintf (stderr, "window_rangemaxabs_variable:  end\n");
01897 #endif
01898 }
01899
01903 static void
01904 window_step_variable ()
01905 {
01906   unsigned int i;
01907 #if DEBUG_INTERFACE
01908   fprintf (stderr, "window_step_variable:  start\n");
01909 #endif
01910   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01911   input->variable[i].step = gtk_spin_button_get_value (window->spin_step);
01912 #if DEBUG_INTERFACE
01913   fprintf (stderr, "window_step_variable:  end\n");
```

```
01914 #endif
01915 }
01916
01920 static void
01921 window_update_variable ()
01922 {
01923   int i;
01924 #if DEBUG_INTERFACE
01925   fprintf (stderr, "window_update_variable:  start\n");
01926 #endif
01927   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01928   if (i < 0)
01929     i = 0;
01930   switch (window_get_algorithm ())
01931     {
01932     case ALGORITHM_SWEEP:
01933     case ALGORITHM_ORTHOGONAL:
01934       input->variable[i].nsweeps
01935         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01936 #if DEBUG_INTERFACE
01937       fprintf (stderr, "window_update_variable:  nsweeps[%d]=%u\n", i,
01938                input->variable[i].nsweeps);
01939 #endif
01940       break;
01941     case ALGORITHM_GENETIC:
01942       input->variable[i].nbits
01943         = gtk_spin_button_get_value_as_int (window->spin_bits);
01944 #if DEBUG_INTERFACE
01945       fprintf (stderr, "window_update_variable:  nbits[%d]=%u\n", i,
01946                input->variable[i].nbits);
01947 #endif
01948     }
01949 #if DEBUG_INTERFACE
01950   fprintf (stderr, "window_update_variable:  end\n");
01951 #endif
01952 }
01953
01959 static int
01960 window_read (char *filename)
01961 {
01962   unsigned int i;
01963 #if DEBUG_INTERFACE
01964   fprintf (stderr, "window_read:  start\n");
01965 #endif
01966
01967   // Reading new input file
01968   input_free ();
01969   input->result = input->variables = NULL;
01970   if (!input_open (filename))
01971     {
01972 #if DEBUG_INTERFACE
01973       fprintf (stderr, "window_read:  end\n");
01974 #endif
01975       return 0;
01976     }
01977
01978   // Setting GTK+ widgets data
01979   gtk_entry_set_text (window->entry_result, input->result);
01980   gtk_entry_set_text (window->entry_variables, input->variables);
01981   gtk_button_set_label (window->button_simulator, input->simulator);
01982   gtk_check_button_set_active (window->check_evaluator,
01983                                (size_t) input->evaluator);
01984   if (input->evaluator)
01985     gtk_button_set_label (window->button_evaluator, input->evaluator);
01986   gtk_check_button_set_active (window->button_algorithm[input->algorithm],
01987                                TRUE);
01988   switch (input->algorithm)
01989     {
01990     case ALGORITHM_MONTE_CARLO:
01991       gtk_spin_button_set_value (window->spin_simulations,
01992                                  (gdouble) input->nsimulations);
01993       // fallthrough
01994     case ALGORITHM_SWEEP:
01995     case ALGORITHM_ORTHOGONAL:
01996       gtk_spin_button_set_value (window->spin_iterations,
01997                                  (gdouble) input->niterations);
01998       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
01999       gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
02000       gtk_check_button_set_active (window->check_climbing, input->nsteps);
02001       if (input->nsteps)
02002         {
02003           gtk_check_button_set_active
02004             (window->button_climbing[input->climbing], TRUE);
02005           gtk_spin_button_set_value (window->spin_steps,
02006                                      (gdouble) input->nsteps);
02007           gtk_spin_button_set_value (window->spin_relaxation,
02008                                      (gdouble) input->relaxation);
```

```
02009           switch (input->climbing)
02010             {
02011             case CLIMBING_METHOD_RANDOM:
02012               gtk_spin_button_set_value (window->spin_estimates,
02013                                          (gdouble) input->nestimates);
02014             }
02015         }
02016       break;
02017     default:
02018       gtk_spin_button_set_value (window->spin_population,
02019                                  (gdouble) input->nsimulations);
02020       gtk_spin_button_set_value (window->spin_generations,
02021                                  (gdouble) input->niterations);
02022       gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
02023       gtk_spin_button_set_value (window->spin_reproduction,
02024                                  input->reproduction_ratio);
02025       gtk_spin_button_set_value (window->spin_adaptation,
02026                                  input->adaptation_ratio);
02027     }
02028   gtk_check_button_set_active (window->button_norm[input->norm], TRUE);
02029   gtk_spin_button_set_value (window->spin_p, input->p);
02030   gtk_spin_button_set_value (window->spin_threshold, input->threshold);
02031   g_signal_handler_block (window->combo_experiment, window->id_experiment);
02032   gtk_combo_box_text_remove_all (window->combo_experiment);
02033   for (i = 0; i < input->nexperiments; ++i)
02034     gtk_combo_box_text_append_text (window->combo_experiment,
02035                                     input->experiment[i].name);
02036   g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
02037   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
02038   g_signal_handler_block (window->combo_variable, window->id_variable);
02039   g_signal_handler_block (window->entry_variable, window->id_variable_label);
02040   gtk_combo_box_text_remove_all (window->combo_variable);
02041   for (i = 0; i < input->nvariables; ++i)
02042     gtk_combo_box_text_append_text (window->combo_variable,
02043                                     input->variable[i].name);
02044   g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
02045   g_signal_handler_unblock (window->combo_variable, window->id_variable);
02046   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02047   window_set_variable ();
02048   window_update ();
02049
02050 #if DEBUG_INTERFACE
02051   fprintf (stderr, "window_read:  end\n");
02052 #endif
02053   return 1;
02054 }
02055
02059 static void
02060 dialog_open_close (GtkFileChooserDialog * dlg,
02062                    int response_id)
02063 {
02064   char *buffer, *directory, *name;
02065
02066 #if DEBUG_INTERFACE
02067   fprintf (stderr, "dialog_open_close:  start\n");
02068 #endif
02069
02070   // Saving a backup of the current input file
02071   directory = g_strdup (input->directory);
02072   name = g_strdup (input->name);
02073
02074   // If OK saving
02075   if (response_id == GTK_RESPONSE_OK)
02076     {
02077
02078       // Traying to open the input file
02079       buffer = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02080       if (!window_read (buffer))
02081         {
02082 #if DEBUG_INTERFACE
02083           fprintf (stderr, "dialog_open_close:  error reading input file\n");
02084 #endif
02085           g_free (buffer);
02086
02087           // Reading backup file on error
02088           buffer = g_build_filename (directory, name, NULL);
02089           input->result = input->variables = NULL;
02090           if (!input_open (buffer))
02091             {
02092
02093               // Closing on backup file reading error
02094 #if DEBUG_INTERFACE
02095               fprintf (stderr,
02096                        "dialog_open_close:  error reading backup file\n");
02097 #endif
02098             }
02099         }
```

```
02100       g_free (buffer);
02101     }
02102
02103   // Freeing and closing
02104   g_free (name);
02105   g_free (directory);
02106   gtk_window_destroy (GTK_WINDOW (dlg));
02107
02108 #if DEBUG_INTERFACE
02109   fprintf (stderr, "dialog_open_close:  end\n");
02110 #endif
02111
02112 }
02113
02117 static void
02118 window_open ()
02119 {
02120   GtkFileChooserDialog *dlg;
02121   GtkFileFilter *filter;
02122
02123 #if DEBUG_INTERFACE
02124   fprintf (stderr, "window_open:  start\n");
02125 #endif
02126
02127   // Opening dialog
02128   dlg = (GtkFileChooserDialog *)
02129     gtk_file_chooser_dialog_new (_("Open input file"),
02130                                  window->window,
02131                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02132                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02133                                  _("_OK"), GTK_RESPONSE_OK, NULL);
02134
02135   // Adding XML filter
02136   filter = (GtkFileFilter *) gtk_file_filter_new ();
02137   gtk_file_filter_set_name (filter, "XML");
02138   gtk_file_filter_add_pattern (filter, "*.xml");
02139   gtk_file_filter_add_pattern (filter, "*.XML");
02140   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02141
02142   // Adding JSON filter
02143   filter = (GtkFileFilter *) gtk_file_filter_new ();
02144   gtk_file_filter_set_name (filter, "JSON");
02145   gtk_file_filter_add_pattern (filter, "*.json");
02146   gtk_file_filter_add_pattern (filter, "*.JSON");
02147   gtk_file_filter_add_pattern (filter, "*.js");
02148   gtk_file_filter_add_pattern (filter, "*.JS");
02149   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02150
02151   // Connecting the close function
02152   g_signal_connect (dlg, "response", G_CALLBACK (dialog_open_close), NULL);
02153
02154   // Showing modal dialog
02155   gtk_window_present (GTK_WINDOW (dlg));
02156
02157 #if DEBUG_INTERFACE
02158   fprintf (stderr, "window_open:  end\n");
02159 #endif
02160 }
02161
02165 static void
02166 dialog_simulator_close (GtkFileChooserDialog * dlg,
02168                         int response_id)
02169 {
02170   GFile *file1, *file2;
02171   char *buffer1, *buffer2;
02172 #if DEBUG_INTERFACE
02173   fprintf (stderr, "dialog_simulator_close:  start\n");
02174 #endif
02175   if (response_id == GTK_RESPONSE_OK)
02176     {
02177       buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02178       file1 = g_file_new_for_path (buffer1);
02179       file2 = g_file_new_for_path (input->directory);
02180       buffer2 = g_file_get_relative_path (file2, file1);
02181       input->simulator = g_strdup (buffer2);
02182       g_free (buffer2);
02183       g_object_unref (file2);
02184       g_object_unref (file1);
02185       g_free (buffer1);
02186     }
02187   gtk_window_destroy (GTK_WINDOW (dlg));
02188 #if DEBUG_INTERFACE
02189   fprintf (stderr, "dialog_simulator_close:  end\n");
02190 #endif
02191 }
02192
02196 static void
```

```
02197 dialog_simulator ()
02198 {
02199   GtkFileChooserDialog *dlg;
02200 #if DEBUG_INTERFACE
02201   fprintf (stderr, "dialog_simulator:  start\n");
02202 #endif
02203   dlg = (GtkFileChooserDialog *)
02204     gtk_file_chooser_dialog_new (_("Open simulator file"),
02205                                  window->window,
02206                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02207                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02208                                  _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02209   g_signal_connect (dlg, "response", G_CALLBACK (dialog_simulator_close), NULL);
02210   gtk_window_present (GTK_WINDOW (dlg));
02211 #if DEBUG_INTERFACE
02212   fprintf (stderr, "dialog_simulator:  end\n");
02213 #endif
02214 }
02215
02219 static void
02220 dialog_evaluator_close (GtkFileChooserDialog * dlg,
02222                         int response_id)
02223 {
02224   GFile *file1, *file2;
02225   char *buffer1, *buffer2;
02226 #if DEBUG_INTERFACE
02227   fprintf (stderr, "dialog_evaluator_close:  start\n");
02228 #endif
02229   if (response_id == GTK_RESPONSE_OK)
02230     {
02231       buffer1 = gtk_file_chooser_get_current_name (GTK_FILE_CHOOSER (dlg));
02232       file1 = g_file_new_for_path (buffer1);
02233       file2 = g_file_new_for_path (input->directory);
02234       buffer2 = g_file_get_relative_path (file2, file1);
02235       input->evaluator = g_strdup (buffer2);
02236       g_free (buffer2);
02237       g_object_unref (file2);
02238       g_object_unref (file1);
02239       g_free (buffer1);
02240     }
02241   gtk_window_destroy (GTK_WINDOW (dlg));
02242 #if DEBUG_INTERFACE
02243   fprintf (stderr, "dialog_evaluator_close:  end\n");
02244 #endif
02245 }
02246
02250 static void
02251 dialog_evaluator ()
02252 {
02253   GtkFileChooserDialog *dlg;
02254 #if DEBUG_INTERFACE
02255   fprintf (stderr, "dialog_evaluator:  start\n");
02256 #endif
02257   dlg = (GtkFileChooserDialog *)
02258     gtk_file_chooser_dialog_new (_("Open evaluator file"),
02259                                  window->window,
02260                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02261                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02262                                  _("_Open"), GTK_RESPONSE_ACCEPT, NULL);
02263   g_signal_connect (dlg, "response", G_CALLBACK (dialog_evaluator_close), NULL);
02264   gtk_window_present (GTK_WINDOW (dlg));
02265 #if DEBUG_INTERFACE
02266   fprintf (stderr, "dialog_evaluator:  end\n");
02267 #endif
02268 }
02269
02273 void
02274 window_new (GtkApplication * application)
02275 {
02276   unsigned int i;
02277   char *buffer, *buffer2, buffer3[64];
02278   const char *label_algorithm[NALGORITHMS] = {
02279     "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02280   };
02281   const char *tip_algorithm[NALGORITHMS] = {
02282     _("Monte-Carlo brute force algorithm"),
02283     _("Sweep brute force algorithm"),
02284     _("Genetic algorithm"),
02285     _("Orthogonal sampling brute force algorithm"),
02286   };
02287   const char *label_climbing[NCLIMBINGS] = {
02288     _("_Coordinates climbing"), _("_Random climbing")
02289   };
02290   const char *tip_climbing[NCLIMBINGS] = {
02291     _("Coordinates climbing estimate method"),
02292     _("Random climbing estimate method")
02293   };
```

```
02294   const char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02295   const char *tip_norm[NNORMS] = {
02296     _("Euclidean error norm (L2)"),
02297     _("Maximum error norm (L)"),
02298     _("P error norm (Lp)"),
02299     _("Taxicab error norm (L1)")
02300   };
02301 #if !GTK4
02302   const char *close = "delete-event";
02303 #else
02304   const char *close = "close-request";
02305 #endif
02306
02307 #if DEBUG_INTERFACE
02308   fprintf (stderr, "window_new:  start\n");
02309 #endif
02310
02311   // Creating the window
02312   window->window = main_window
02313     = (GtkWindow *) gtk_application_window_new (application);
02314
02315   // Finish when closing the window
02316   g_signal_connect_swapped (window->window, close,
02317                             G_CALLBACK (g_application_quit),
02318                             G_APPLICATION (application));
02319
02320   // Setting the window title
02321   gtk_window_set_title (window->window, "MPCOTool");
02322
02323   // Creating the open button
02324   window->button_open = (GtkButton *)
02325 #if !GTK4
02326     gtk_button_new_from_icon_name ("document-open", GTK_ICON_SIZE_BUTTON);
02327 #else
02328     gtk_button_new_from_icon_name ("document-open");
02329 #endif
02330   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02331
02332   // Creating the save button
02333   window->button_save = (GtkButton *)
02334 #if !GTK4
02335     gtk_button_new_from_icon_name ("document-save", GTK_ICON_SIZE_BUTTON);
02336 #else
02337     gtk_button_new_from_icon_name ("document-save");
02338 #endif
02339   g_signal_connect (window->button_save, "clicked", (GCallback) window_save,
02340                     NULL);
02341
02342   // Creating the run button
02343   window->button_run = (GtkButton *)
02344 #if !GTK4
02345     gtk_button_new_from_icon_name ("system-run", GTK_ICON_SIZE_BUTTON);
02346 #else
02347     gtk_button_new_from_icon_name ("system-run");
02348 #endif
02349   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02350
02351   // Creating the options button
02352   window->button_options = (GtkButton *)
02353 #if !GTK4
02354     gtk_button_new_from_icon_name ("preferences-system", GTK_ICON_SIZE_BUTTON);
02355 #else
02356     gtk_button_new_from_icon_name ("preferences-system");
02357 #endif
02358   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02359
02360   // Creating the help button
02361   window->button_help = (GtkButton *)
02362 #if !GTK4
02363     gtk_button_new_from_icon_name ("help-browser", GTK_ICON_SIZE_BUTTON);
02364 #else
02365     gtk_button_new_from_icon_name ("help-browser");
02366 #endif
02367   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02368
02369   // Creating the about button
02370   window->button_about = (GtkButton *)
02371 #if !GTK4
02372     gtk_button_new_from_icon_name ("help-about", GTK_ICON_SIZE_BUTTON);
02373 #else
02374     gtk_button_new_from_icon_name ("help-about");
02375 #endif
02376   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02377
02378   // Creating the exit button
02379   window->button_exit = (GtkButton *)
02380 #if !GTK4
```

```
02381     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
02382 #else
02383     gtk_button_new_from_icon_name ("application-exit");
02384 #endif
02385   g_signal_connect_swapped (window->button_exit, "clicked",
02386                             G_CALLBACK (g_application_quit),
02387                             G_APPLICATION (application));
02388
02389   // Creating the buttons bar
02390   window->box_buttons = (GtkBox *) gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
02391   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_open));
02392   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_save));
02393   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_run));
02394   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_options));
02395   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_help));
02396   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_about));
02397   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_exit));
02398
02399   // Creating the simulator program label and entry
02400   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02401   window->button_simulator = (GtkButton *)
02402     gtk_button_new_with_mnemonic (_("Simulator program"));
02403   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02404                                _("Simulator program executable file"));
02405   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02406   g_signal_connect (window->button_simulator, "clicked",
02407                     G_CALLBACK (dialog_simulator), NULL);
02408
02409   // Creating the evaluator program label and entry
02410   window->check_evaluator = (GtkCheckButton *)
02411     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02412   g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
02413   window->button_evaluator = (GtkButton *)
02414     gtk_button_new_with_mnemonic (_("Evaluator program"));
02415   gtk_widget_set_tooltip_text
02416     (GTK_WIDGET (window->button_evaluator),
02417      _("Optional evaluator program executable file"));
02418   g_signal_connect (window->button_evaluator, "clicked",
02419                     G_CALLBACK (dialog_evaluator), NULL);
02420
02421   // Creating the results files labels and entries
02422   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02423   window->entry_result = (GtkEntry *) gtk_entry_new ();
02424   gtk_widget_set_tooltip_text
02425     (GTK_WIDGET (window->entry_result), _("Best results file"));
02426   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02427   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02428   gtk_widget_set_tooltip_text
02429     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02430
02431   // Creating the files grid and attaching widgets
02432   window->grid_files = (GtkGrid *) gtk_grid_new ();
02433   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
02434                    0, 0, 1, 1);
02435   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
02436                    1, 0, 1, 1);
02437   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
02438                    0, 1, 1, 1);
02439   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
02440                    1, 1, 1, 1);
02441   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
02442                    0, 2, 1, 1);
02443   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
02444                    1, 2, 1, 1);
02445   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
02446                    0, 3, 1, 1);
02447   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
02448                    1, 3, 1, 1);
02449
02450   // Creating the algorithm properties
02451   window->label_simulations = (GtkLabel *) gtk_label_new
02452     (_("Simulations number"));
02453   window->spin_simulations
02454     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02455   gtk_widget_set_tooltip_text
02456     (GTK_WIDGET (window->spin_simulations),
02457      _("Number of simulations to perform for each iteration"));
02458   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02459   window->label_iterations = (GtkLabel *)
02460     gtk_label_new (_("Iterations number"));
02461   window->spin_iterations
02462     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02463   gtk_widget_set_tooltip_text
02464     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02465   g_signal_connect
02466     (window->spin_iterations, "value-changed", window_update, NULL);
02467   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
```

```
02468   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02469   window->spin_tolerance =
02470     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02471   gtk_widget_set_tooltip_text
02472     (GTK_WIDGET (window->spin_tolerance),
02473      _("Tolerance to set the variable interval on the next iteration"));
02474   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02475   window->spin_bests
02476     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02477   gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->spin_bests),
02479      _("Number of best simulations used to set the variable interval "
02480        "on the next iteration"));
02481   window->label_population
02482     = (GtkLabel *) gtk_label_new (_("Population number"));
02483   window->spin_population
02484     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02485   gtk_widget_set_tooltip_text
02486     (GTK_WIDGET (window->spin_population),
02487      _("Number of population for the genetic algorithm"));
02488   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02489   window->label_generations
02490     = (GtkLabel *) gtk_label_new (_("Generations number"));
02491   window->spin_generations
02492     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02493   gtk_widget_set_tooltip_text
02494     (GTK_WIDGET (window->spin_generations),
02495      _("Number of generations for the genetic algorithm"));
02496   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02497   window->spin_mutation
02498     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02499   gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_mutation),
02501      _("Ratio of mutation for the genetic algorithm"));
02502   window->label_reproduction
02503     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02504   window->spin_reproduction
02505     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02506   gtk_widget_set_tooltip_text
02507     (GTK_WIDGET (window->spin_reproduction),
02508      _("Ratio of reproduction for the genetic algorithm"));
02509   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02510   window->spin_adaptation
02511     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02512   gtk_widget_set_tooltip_text
02513     (GTK_WIDGET (window->spin_adaptation),
02514      _("Ratio of adaptation for the genetic algorithm"));
02515   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02516   window->spin_threshold = (GtkSpinButton *)
02517     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02518                                     precision[DEFAULT_PRECISION]);
02519   gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_threshold),
02521      _("Threshold in the objective function to finish the simulations"));
02522   window->scrolled_threshold = (GtkScrolledWindow *)
02523 #if !GTK4
02524     gtk_scrolled_window_new (NULL, NULL);
02525 #else
02526     gtk_scrolled_window_new ();
02527 #endif
02528   gtk_scrolled_window_set_child (window->scrolled_threshold,
02529                                  GTK_WIDGET (window->spin_threshold));
02530 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02531 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02532 //                         GTK_ALIGN_FILL);
02533
02534   // Creating the hill climbing method properties
02535   window->check_climbing = (GtkCheckButton *)
02536     gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
02537   g_signal_connect (window->check_climbing, "toggled", window_update, NULL);
02538   window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02539 #if !GTK4
02540   window->button_climbing[0] = (GtkRadioButton *)
02541     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02542 #else
02543   window->button_climbing[0] = (GtkCheckButton *)
02544     gtk_check_button_new_with_mnemonic (label_climbing[0]);
02545 #endif
02546   gtk_grid_attach (window->grid_climbing,
02547                    GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02548   g_signal_connect (window->button_climbing[0], "toggled", window_update, NULL);
02549   for (i = 0; ++i < NCLIMBINGS;)
02550     {
02551 #if !GTK4
02552       window->button_climbing[i] = (GtkRadioButton *)
02553         gtk_radio_button_new_with_mnemonic
02554         (gtk_radio_button_get_group (window->button_climbing[0]),
```

```
02555           label_climbing[i]);
02556 #else
02557       window->button_climbing[i] = (GtkCheckButton *)
02558         gtk_check_button_new_with_mnemonic (label_climbing[i]);
02559       gtk_check_button_set_group (window->button_climbing[i],
02560                                   window->button_climbing[0]);
02561 #endif
02562       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02563                                    tip_climbing[i]);
02564       gtk_grid_attach (window->grid_climbing,
02565                        GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02566       g_signal_connect (window->button_climbing[i], "toggled", window_update,
02567                         NULL);
02568     }
02569   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02570   window->spin_steps = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02572   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02573   window->label_estimates
02574     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02575   window->spin_estimates = (GtkSpinButton *)
02576     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02577   window->label_relaxation
02578     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02579   window->spin_relaxation = (GtkSpinButton *)
02580     gtk_spin_button_new_with_range (0., 2., 0.001);
02581   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
02582                    0, NCLIMBINGS, 1, 1);
02583   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
02584                    1, NCLIMBINGS, 1, 1);
02585   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
02586                    0, NCLIMBINGS + 1, 1, 1);
02587   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
02588                    1, NCLIMBINGS + 1, 1, 1);
02589   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
02590                    0, NCLIMBINGS + 2, 1, 1);
02591   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
02592                    1, NCLIMBINGS + 2, 1, 1);
02593
02594   // Creating the array of algorithms
02595   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02596 #if !GTK4
02597   window->button_algorithm[0] = (GtkRadioButton *)
02598     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02599 #else
02600   window->button_algorithm[0] = (GtkCheckButton *)
02601     gtk_check_button_new_with_mnemonic (label_algorithm[0]);
02602 #endif
02603   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02604                                tip_algorithm[0]);
02605   gtk_grid_attach (window->grid_algorithm,
02606                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02607   g_signal_connect (window->button_algorithm[0], "toggled",
02608                     window_set_algorithm, NULL);
02609   for (i = 0; ++i < NALGORITHMS;)
02610     {
02611 #if !GTK4
02612       window->button_algorithm[i] = (GtkRadioButton *)
02613         gtk_radio_button_new_with_mnemonic
02614         (gtk_radio_button_get_group (window->button_algorithm[0]),
02615          label_algorithm[i]);
02616 #else
02617       window->button_algorithm[i] = (GtkCheckButton *)
02618         gtk_check_button_new_with_mnemonic (label_algorithm[i]);
02619       gtk_check_button_set_group (window->button_algorithm[i],
02620                                   window->button_algorithm[0]),
02621 #endif
02622         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02623                                      tip_algorithm[i]);
02624       gtk_grid_attach (window->grid_algorithm,
02625                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02626       g_signal_connect (window->button_algorithm[i], "toggled",
02627                         window_set_algorithm, NULL);
02628     }
02629   gtk_grid_attach (window->grid_algorithm,
02630                    GTK_WIDGET (window->label_simulations),
02631                    0, NALGORITHMS, 1, 1);
02632   gtk_grid_attach (window->grid_algorithm,
02633                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02634   gtk_grid_attach (window->grid_algorithm,
02635                    GTK_WIDGET (window->label_iterations),
02636                    0, NALGORITHMS + 1, 1, 1);
02637   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
02638                    1, NALGORITHMS + 1, 1, 1);
02639   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
02640                    0, NALGORITHMS + 2, 1, 1);
02641   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
```

```
02642                    1, NALGORITHMS + 2, 1, 1);
02643   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
02644                    0, NALGORITHMS + 3, 1, 1);
02645   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
02646                    1, NALGORITHMS + 3, 1, 1);
02647   gtk_grid_attach (window->grid_algorithm,
02648                    GTK_WIDGET (window->label_population),
02649                    0, NALGORITHMS + 4, 1, 1);
02650   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
02651                    1, NALGORITHMS + 4, 1, 1);
02652   gtk_grid_attach (window->grid_algorithm,
02653                    GTK_WIDGET (window->label_generations),
02654                    0, NALGORITHMS + 5, 1, 1);
02655   gtk_grid_attach (window->grid_algorithm,
02656                    GTK_WIDGET (window->spin_generations),
02657                    1, NALGORITHMS + 5, 1, 1);
02658   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
02659                    0, NALGORITHMS + 6, 1, 1);
02660   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
02661                    1, NALGORITHMS + 6, 1, 1);
02662   gtk_grid_attach (window->grid_algorithm,
02663                    GTK_WIDGET (window->label_reproduction),
02664                    0, NALGORITHMS + 7, 1, 1);
02665   gtk_grid_attach (window->grid_algorithm,
02666                    GTK_WIDGET (window->spin_reproduction),
02667                    1, NALGORITHMS + 7, 1, 1);
02668   gtk_grid_attach (window->grid_algorithm,
02669                    GTK_WIDGET (window->label_adaptation),
02670                    0, NALGORITHMS + 8, 1, 1);
02671   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),
02672                    1, NALGORITHMS + 8, 1, 1);
02673   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
02674                    0, NALGORITHMS + 9, 2, 1);
02675   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
02676                    0, NALGORITHMS + 10, 2, 1);
02677   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
02678                    0, NALGORITHMS + 11, 1, 1);
02679   gtk_grid_attach (window->grid_algorithm,
02680                    GTK_WIDGET (window->scrolled_threshold),
02681                    1, NALGORITHMS + 11, 1, 1);
02682   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02683   gtk_frame_set_child (window->frame_algorithm,
02684                        GTK_WIDGET (window->grid_algorithm));
02685
02686   // Creating the variable widgets
02687   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02688   gtk_widget_set_tooltip_text
02689     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02690   window->id_variable = g_signal_connect
02691     (window->combo_variable, "changed", window_set_variable, NULL);
02692 #if !GTK4
02693   window->button_add_variable = (GtkButton *)
02694     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02695 #else
02696   window->button_add_variable = (GtkButton *)
02697     gtk_button_new_from_icon_name ("list-add");
02698 #endif
02699   g_signal_connect (window->button_add_variable, "clicked", window_add_variable,
02700                     NULL);
02701   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02702                                _("Add variable"));
02703 #if !GTK4
02704   window->button_remove_variable = (GtkButton *)
02705     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02706 #else
02707   window->button_remove_variable = (GtkButton *)
02708     gtk_button_new_from_icon_name ("list-remove");
02709 #endif
02710   g_signal_connect (window->button_remove_variable, "clicked",
02711                     window_remove_variable, NULL);
02712   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02713                                _("Remove variable"));
02714   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02715   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02716   gtk_widget_set_tooltip_text
02717     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02718   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02719   window->id_variable_label = g_signal_connect
02720     (window->entry_variable, "changed", window_label_variable, NULL);
02721   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02722   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02723     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02724   gtk_widget_set_tooltip_text
02725     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02726   window->scrolled_min = (GtkScrolledWindow *)
02727 #if !GTK4
02728     gtk_scrolled_window_new (NULL, NULL);
```

```
02729 #else
02730     gtk_scrolled_window_new ();
02731 #endif
02732   gtk_scrolled_window_set_child (window->scrolled_min,
02733                                  GTK_WIDGET (window->spin_min));
02734   g_signal_connect (window->spin_min, "value-changed",
02735                     window_rangemin_variable, NULL);
02736   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02737   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02738     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02739   gtk_widget_set_tooltip_text
02740     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02741   window->scrolled_max = (GtkScrolledWindow *)
02742 #if !GTK4
02743     gtk_scrolled_window_new (NULL, NULL);
02744 #else
02745     gtk_scrolled_window_new ();
02746 #endif
02747   gtk_scrolled_window_set_child (window->scrolled_max,
02748                                  GTK_WIDGET (window->spin_max));
02749   g_signal_connect (window->spin_max, "value-changed",
02750                     window_rangemax_variable, NULL);
02751   window->check_minabs = (GtkCheckButton *)
02752     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02753   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02754   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02755     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02756   gtk_widget_set_tooltip_text
02757     (GTK_WIDGET (window->spin_minabs),
02758      _("Minimum allowed value of the variable"));
02759   window->scrolled_minabs = (GtkScrolledWindow *)
02760 #if !GTK4
02761     gtk_scrolled_window_new (NULL, NULL);
02762 #else
02763     gtk_scrolled_window_new ();
02764 #endif
02765   gtk_scrolled_window_set_child (window->scrolled_minabs,
02766                                  GTK_WIDGET (window->spin_minabs));
02767   g_signal_connect (window->spin_minabs, "value-changed",
02768                     window_rangeminabs_variable, NULL);
02769   window->check_maxabs = (GtkCheckButton *)
02770     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02771   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02772   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02773     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02774   gtk_widget_set_tooltip_text
02775     (GTK_WIDGET (window->spin_maxabs),
02776      _("Maximum allowed value of the variable"));
02777   window->scrolled_maxabs = (GtkScrolledWindow *)
02778 #if !GTK4
02779     gtk_scrolled_window_new (NULL, NULL);
02780 #else
02781     gtk_scrolled_window_new ();
02782 #endif
02783   gtk_scrolled_window_set_child (window->scrolled_maxabs,
02784                                  GTK_WIDGET (window->spin_maxabs));
02785   g_signal_connect (window->spin_maxabs, "value-changed",
02786                     window_rangemaxabs_variable, NULL);
02787   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02788   window->spin_precision = (GtkSpinButton *)
02789     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02790   gtk_widget_set_tooltip_text
02791     (GTK_WIDGET (window->spin_precision),
02792      _("Number of precision floating point digits\n"
02793        "0 is for integer numbers"));
02794   g_signal_connect (window->spin_precision, "value-changed",
02795                     window_precision_variable, NULL);
02796   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02797   window->spin_sweeps =
02798     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02799   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02800                                _("Number of steps sweeping the variable"));
02801   g_signal_connect (window->spin_sweeps, "value-changed",
02802                     window_update_variable, NULL);
02803   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02804   window->spin_bits
02805     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02806   gtk_widget_set_tooltip_text
02807     (GTK_WIDGET (window->spin_bits),
02808      _("Number of bits to encode the variable"));
02809   g_signal_connect
02810     (window->spin_bits, "value-changed", window_update_variable, NULL);
02811   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02812   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02813     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02814   gtk_widget_set_tooltip_text
02815     (GTK_WIDGET (window->spin_step),
```

```
02816       _("Initial step size for the hill climbing method"));
02817   window->scrolled_step = (GtkScrolledWindow *)
02818 #if !GTK4
02819     gtk_scrolled_window_new (NULL, NULL);
02820 #else
02821     gtk_scrolled_window_new ();
02822 #endif
02823   gtk_scrolled_window_set_child (window->scrolled_step,
02824                                  GTK_WIDGET (window->spin_step));
02825   g_signal_connect
02826     (window->spin_step, "value-changed", window_step_variable, NULL);
02827   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02828   gtk_grid_attach (window->grid_variable,
02829                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02830   gtk_grid_attach (window->grid_variable,
02831                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02832   gtk_grid_attach (window->grid_variable,
02833                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02834   gtk_grid_attach (window->grid_variable,
02835                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02836   gtk_grid_attach (window->grid_variable,
02837                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02838   gtk_grid_attach (window->grid_variable,
02839                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02840   gtk_grid_attach (window->grid_variable,
02841                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02842   gtk_grid_attach (window->grid_variable,
02843                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02844   gtk_grid_attach (window->grid_variable,
02845                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02846   gtk_grid_attach (window->grid_variable,
02847                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02848   gtk_grid_attach (window->grid_variable,
02849                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02850   gtk_grid_attach (window->grid_variable,
02851                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02852   gtk_grid_attach (window->grid_variable,
02853                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02854   gtk_grid_attach (window->grid_variable,
02855                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02856   gtk_grid_attach (window->grid_variable,
02857                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02858   gtk_grid_attach (window->grid_variable,
02859                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02860   gtk_grid_attach (window->grid_variable,
02861                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02862   gtk_grid_attach (window->grid_variable,
02863                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02864   gtk_grid_attach (window->grid_variable,
02865                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02866   gtk_grid_attach (window->grid_variable,
02867                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02868   gtk_grid_attach (window->grid_variable,
02869                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02870   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02871   gtk_frame_set_child (window->frame_variable,
02872                        GTK_WIDGET (window->grid_variable));
02873
02874   // Creating the experiment widgets
02875   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02876   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02877                                _("Experiment selector"));
02878   window->id_experiment = g_signal_connect
02879     (window->combo_experiment, "changed", window_set_experiment, NULL);
02880 #if !GTK4
02881   window->button_add_experiment = (GtkButton *)
02882     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02883 #else
02884   window->button_add_experiment = (GtkButton *)
02885     gtk_button_new_from_icon_name ("list-add");
02886 #endif
02887   g_signal_connect
02888     (window->button_add_experiment, "clicked", window_add_experiment, NULL);
02889   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02890                                _("Add experiment"));
02891 #if !GTK4
02892   window->button_remove_experiment = (GtkButton *)
02893     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02894 #else
02895   window->button_remove_experiment = (GtkButton *)
02896     gtk_button_new_from_icon_name ("list-remove");
02897 #endif
02898   g_signal_connect (window->button_remove_experiment, "clicked",
02899                     window_remove_experiment, NULL);
02900   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02901                                _("Remove experiment"));
02902   window->label_experiment
```

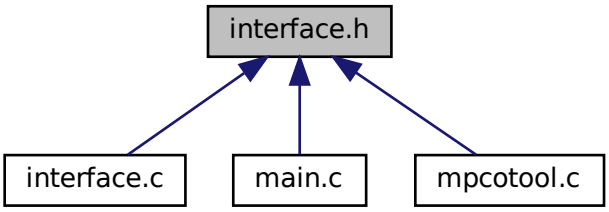
```
02903     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02904   window->button_experiment = (GtkButton *)
02905     gtk_button_new_with_mnemonic (_("Experimental data file"));
02906   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02907                                _("Experimental data file"));
02908   g_signal_connect (window->button_experiment, "clicked",
02909                     window_name_experiment, NULL);
02910   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02911   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02912   window->spin_weight
02913     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02914   gtk_widget_set_tooltip_text
02915     (GTK_WIDGET (window->spin_weight),
02916      _("Weight factor to build the objective function"));
02917   g_signal_connect
02918     (window->spin_weight, "value-changed", window_weight_experiment, NULL);
02919   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02920   gtk_grid_attach (window->grid_experiment,
02921                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02922   gtk_grid_attach (window->grid_experiment,
02923                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02924   gtk_grid_attach (window->grid_experiment,
02925                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02926   gtk_grid_attach (window->grid_experiment,
02927                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02928   gtk_grid_attach (window->grid_experiment,
02929                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02930   gtk_grid_attach (window->grid_experiment,
02931                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02932   gtk_grid_attach (window->grid_experiment,
02933                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02934   for (i = 0; i < MAX_NINPUTS; ++i)
02935     {
02936       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02937       window->check_template[i] = (GtkCheckButton *)
02938         gtk_check_button_new_with_label (buffer3);
02939       window->id_template[i]
02940         = g_signal_connect (window->check_template[i], "toggled",
02941                             window_inputs_experiment, NULL);
02942       gtk_grid_attach (window->grid_experiment,
02943                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02944       window->button_template[i] = (GtkButton *)
02945         gtk_button_new_with_mnemonic (_("Input template"));
02946
02947       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02948                                    _("Experimental input template file"));
02949       window->id_input[i] =
02950         g_signal_connect_swapped (window->button_template[i], "clicked",
02951                                   (GCallback) window_template_experiment,
02952                                   (void *) (size_t) i);
02953       gtk_grid_attach (window->grid_experiment,
02954                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02955     }
02956   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02957   gtk_frame_set_child (window->frame_experiment,
02958                        GTK_WIDGET (window->grid_experiment));
02959
02960   // Creating the error norm widgets
02961   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02962   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02963   gtk_frame_set_child (window->frame_norm, GTK_WIDGET (window->grid_norm));
02964 #if !GTK4
02965   window->button_norm[0] = (GtkRadioButton *)
02966     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02967 #else
02968   window->button_norm[0] = (GtkCheckButton *)
02969     gtk_check_button_new_with_mnemonic (label_norm[0]);
02970 #endif
02971   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02972                                tip_norm[0]);
02973   gtk_grid_attach (window->grid_norm,
02974                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02975   g_signal_connect (window->button_norm[0], "toggled", window_update, NULL);
02976   for (i = 0; ++i < NNORMS;)
02977     {
02978 #if !GTK4
02979       window->button_norm[i] = (GtkRadioButton *)
02980         gtk_radio_button_new_with_mnemonic
02981         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02982 #else
02983       window->button_norm[i] = (GtkCheckButton *)
02984         gtk_check_button_new_with_mnemonic (label_norm[i]);
02985       gtk_check_button_set_group (window->button_norm[i],
02986                                   window->button_norm[0]);
02987 #endif
02988       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02989                                    tip_norm[i]);
```

```
02990       gtk_grid_attach (window->grid_norm,
02991                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02992       g_signal_connect (window->button_norm[i], "toggled", window_update, NULL);
02993     }
02994   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02995   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02996   window->spin_p = (GtkSpinButton *)
02997     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02998   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02999                                _("P parameter for the P error norm"));
03000   window->scrolled_p = (GtkScrolledWindow *)
03001 #if !GTK4
03002     gtk_scrolled_window_new (NULL, NULL);
03003 #else
03004     gtk_scrolled_window_new ();
03005 #endif
03006   gtk_scrolled_window_set_child (window->scrolled_p,
03007                                  GTK_WIDGET (window->spin_p));
03008   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
03009   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
03010   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
03011                    1, 2, 1, 2);
03012
03013   // Creating the grid and attaching the widgets to the grid
03014   window->grid = (GtkGrid *) gtk_grid_new ();
03015   gtk_grid_attach (window->grid, GTK_WIDGET (window->box_buttons), 0, 0, 3, 1);
03016   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
03017   gtk_grid_attach (window->grid,
03018                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
03019   gtk_grid_attach (window->grid,
03020                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
03021   gtk_grid_attach (window->grid,
03022                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
03023   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
03024   gtk_window_set_child (window->window, GTK_WIDGET (window->grid));
03025
03026   // Setting the window logo
03027   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03028 #if !GTK4
03029   gtk_window_set_icon (window->window, window->logo);
03030 #endif
03031
03032   // Showing the window
03033 #if !GTK4
03034   gtk_widget_show_all (GTK_WIDGET (window->window));
03035 #else
03036   gtk_widget_show (GTK_WIDGET (window->window));
03037 #endif
03038
03039   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
03040 #if GTK_MINOR_VERSION >= 16
03041   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03042   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03043   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03044   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03045   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
03046   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
03047   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
03048 #endif
03049
03050   // Reading initial example
03051   input_new ();
03052   buffer2 = g_get_current_dir ();
03053   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03054   g_free (buffer2);
03055   window_read (buffer);
03056   g_free (buffer);
03057
03058 #if DEBUG_INTERFACE
03059   fprintf (stderr, "window_new:  start\n");
03060 #endif
03061 }
```

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

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

application	GtkApplication struct.

Definition at line 2274 of file interface.c.

```
02275 {
02276   unsigned int i;
02277   char *buffer, *buffer2, buffer3[64];
02278   const char *label_algorithm[NALGORITHMS] = {
02279     "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02280   };
02281   const char *tip_algorithm[NALGORITHMS] = {
02282     _("Monte-Carlo brute force algorithm"),
02283     _("Sweep brute force algorithm"),
02284     _("Genetic algorithm"),
02285     _("Orthogonal sampling brute force algorithm"),
02286   };
02287   const char *label_climbing[NCLIMBINGS] = {
02288     _("_Coordinates climbing"), _("_Random climbing")
02289   };
02290   const char *tip_climbing[NCLIMBINGS] = {
02291     _("Coordinates climbing estimate method"),
02292     _("Random climbing estimate method")
02293   };
02294   const char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02295   const char *tip_norm[NNORMS] = {
02296     _("Euclidean error norm (L2)"),
02297     _("Maximum error norm (L)"),
02298     _("P error norm (Lp)"),
02299     _("Taxicab error norm (L1)")
02300   };
02301 #if !GTK4
02302   const char *close = "delete-event";
02303 #else
02304   const char *close = "close-request";
02305 #endif
02306
02307 #if DEBUG_INTERFACE
```

```
02308   fprintf (stderr, "window_new:  start\n");
02309 #endif
02310
02311   // Creating the window
02312   window->window = main_window
02313     = (GtkWindow *) gtk_application_window_new (application);
02314
02315   // Finish when closing the window
02316   g_signal_connect_swapped (window->window, close,
02317                             G_CALLBACK (g_application_quit),
02318                             G_APPLICATION (application));
02319
02320   // Setting the window title
02321   gtk_window_set_title (window->window, "MPCOTool");
02322
02323   // Creating the open button
02324   window->button_open = (GtkButton *)
02325 #if !GTK4
02326     gtk_button_new_from_icon_name ("document-open", GTK_ICON_SIZE_BUTTON);
02327 #else
02328     gtk_button_new_from_icon_name ("document-open");
02329 #endif
02330   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02331
02332   // Creating the save button
02333   window->button_save = (GtkButton *)
02334 #if !GTK4
02335     gtk_button_new_from_icon_name ("document-save", GTK_ICON_SIZE_BUTTON);
02336 #else
02337     gtk_button_new_from_icon_name ("document-save");
02338 #endif
02339   g_signal_connect (window->button_save, "clicked", (GCallback) window_save,
02340                     NULL);
02341
02342   // Creating the run button
02343   window->button_run = (GtkButton *)
02344 #if !GTK4
02345     gtk_button_new_from_icon_name ("system-run", GTK_ICON_SIZE_BUTTON);
02346 #else
02347     gtk_button_new_from_icon_name ("system-run");
02348 #endif
02349   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02350
02351   // Creating the options button
02352   window->button_options = (GtkButton *)
02353 #if !GTK4
02354     gtk_button_new_from_icon_name ("preferences-system", GTK_ICON_SIZE_BUTTON);
02355 #else
02356     gtk_button_new_from_icon_name ("preferences-system");
02357 #endif
02358   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02359
02360   // Creating the help button
02361   window->button_help = (GtkButton *)
02362 #if !GTK4
02363     gtk_button_new_from_icon_name ("help-browser", GTK_ICON_SIZE_BUTTON);
02364 #else
02365     gtk_button_new_from_icon_name ("help-browser");
02366 #endif
02367   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02368
02369   // Creating the about button
02370   window->button_about = (GtkButton *)
02371 #if !GTK4
02372     gtk_button_new_from_icon_name ("help-about", GTK_ICON_SIZE_BUTTON);
02373 #else
02374     gtk_button_new_from_icon_name ("help-about");
02375 #endif
02376   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02377
02378   // Creating the exit button
02379   window->button_exit = (GtkButton *)
02380 #if !GTK4
02381     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
02382 #else
02383     gtk_button_new_from_icon_name ("application-exit");
02384 #endif
02385   g_signal_connect_swapped (window->button_exit, "clicked",
02386                             G_CALLBACK (g_application_quit),
02387                             G_APPLICATION (application));
02388
02389   // Creating the buttons bar
02390   window->box_buttons = (GtkBox *) gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
02391   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_open));
02392   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_save));
02393   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_run));
02394   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_options));
```

```
02395   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_help));
02396   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_about));
02397   gtk_box_append (window->box_buttons, GTK_WIDGET (window->button_exit));
02398
02399   // Creating the simulator program label and entry
02400   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02401   window->button_simulator = (GtkButton *)
02402     gtk_button_new_with_mnemonic (_("Simulator program"));
02403   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02404                                _("Simulator program executable file"));
02405   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02406   g_signal_connect (window->button_simulator, "clicked",
02407                     G_CALLBACK (dialog_simulator), NULL);
02408
02409   // Creating the evaluator program label and entry
02410   window->check_evaluator = (GtkCheckButton *)
02411     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02412   g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
02413   window->button_evaluator = (GtkButton *)
02414     gtk_button_new_with_mnemonic (_("Evaluator program"));
02415   gtk_widget_set_tooltip_text
02416     (GTK_WIDGET (window->button_evaluator),
02417      _("Optional evaluator program executable file"));
02418   g_signal_connect (window->button_evaluator, "clicked",
02419                     G_CALLBACK (dialog_evaluator), NULL);
02420
02421   // Creating the results files labels and entries
02422   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02423   window->entry_result = (GtkEntry *) gtk_entry_new ();
02424   gtk_widget_set_tooltip_text
02425     (GTK_WIDGET (window->entry_result), _("Best results file"));
02426   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02427   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02428   gtk_widget_set_tooltip_text
02429     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02430
02431   // Creating the files grid and attaching widgets
02432   window->grid_files = (GtkGrid *) gtk_grid_new ();
02433   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
02434                    0, 0, 1, 1);
02435   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
02436                    1, 0, 1, 1);
02437   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
02438                    0, 1, 1, 1);
02439   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
02440                    1, 1, 1, 1);
02441   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
02442                    0, 2, 1, 1);
02443   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
02444                    1, 2, 1, 1);
02445   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
02446                    0, 3, 1, 1);
02447   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
02448                    1, 3, 1, 1);
02449
02450   // Creating the algorithm properties
02451   window->label_simulations = (GtkLabel *) gtk_label_new
02452     (_("Simulations number"));
02453   window->spin_simulations
02454     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02455   gtk_widget_set_tooltip_text
02456     (GTK_WIDGET (window->spin_simulations),
02457      _("Number of simulations to perform for each iteration"));
02458   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02459   window->label_iterations = (GtkLabel *)
02460     gtk_label_new (_("Iterations number"));
02461   window->spin_iterations
02462     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02463   gtk_widget_set_tooltip_text
02464     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02465   g_signal_connect
02466     (window->spin_iterations, "value-changed", window_update, NULL);
02467   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02468   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02469   window->spin_tolerance =
02470     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02471   gtk_widget_set_tooltip_text
02472     (GTK_WIDGET (window->spin_tolerance),
02473      _("Tolerance to set the variable interval on the next iteration"));
02474   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02475   window->spin_bests
02476     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02477   gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->spin_bests),
02479      _("Number of best simulations used to set the variable interval "
02480        "on the next iteration"));
02481   window->label_population
```

```
02482     = (GtkLabel *) gtk_label_new (_("Population number"));
02483   window->spin_population
02484     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02485   gtk_widget_set_tooltip_text
02486     (GTK_WIDGET (window->spin_population),
02487      _("Number of population for the genetic algorithm"));
02488   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02489   window->label_generations
02490     = (GtkLabel *) gtk_label_new (_("Generations number"));
02491   window->spin_generations
02492     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02493   gtk_widget_set_tooltip_text
02494     (GTK_WIDGET (window->spin_generations),
02495      _("Number of generations for the genetic algorithm"));
02496   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02497   window->spin_mutation
02498     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02499   gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_mutation),
02501      _("Ratio of mutation for the genetic algorithm"));
02502   window->label_reproduction
02503     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02504   window->spin_reproduction
02505     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02506   gtk_widget_set_tooltip_text
02507     (GTK_WIDGET (window->spin_reproduction),
02508      _("Ratio of reproduction for the genetic algorithm"));
02509   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02510   window->spin_adaptation
02511     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02512   gtk_widget_set_tooltip_text
02513     (GTK_WIDGET (window->spin_adaptation),
02514      _("Ratio of adaptation for the genetic algorithm"));
02515   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02516   window->spin_threshold = (GtkSpinButton *)
02517     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02518                                     precision[DEFAULT_PRECISION]);
02519   gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_threshold),
02521      _("Threshold in the objective function to finish the simulations"));
02522   window->scrolled_threshold = (GtkScrolledWindow *)
02523 #if !GTK4
02524     gtk_scrolled_window_new (NULL, NULL);
02525 #else
02526     gtk_scrolled_window_new ();
02527 #endif
02528   gtk_scrolled_window_set_child (window->scrolled_threshold,
02529                                  GTK_WIDGET (window->spin_threshold));
02530 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02531 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02532 //                         GTK_ALIGN_FILL);
02533
02534   // Creating the hill climbing method properties
02535   window->check_climbing = (GtkCheckButton *)
02536     gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
02537   g_signal_connect (window->check_climbing, "toggled", window_update, NULL);
02538   window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02539 #if !GTK4
02540   window->button_climbing[0] = (GtkRadioButton *)
02541     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02542 #else
02543   window->button_climbing[0] = (GtkCheckButton *)
02544     gtk_check_button_new_with_mnemonic (label_climbing[0]);
02545 #endif
02546   gtk_grid_attach (window->grid_climbing,
02547                    GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02548   g_signal_connect (window->button_climbing[0], "toggled", window_update, NULL);
02549   for (i = 0; ++i < NCLIMBINGS;)
02550     {
02551 #if !GTK4
02552       window->button_climbing[i] = (GtkRadioButton *)
02553         gtk_radio_button_new_with_mnemonic
02554         (gtk_radio_button_get_group (window->button_climbing[0]),
02555          label_climbing[i]);
02556 #else
02557       window->button_climbing[i] = (GtkCheckButton *)
02558         gtk_check_button_new_with_mnemonic (label_climbing[i]);
02559       gtk_check_button_set_group (window->button_climbing[i],
02560                                   window->button_climbing[0]);
02561 #endif
02562       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02563                                    tip_climbing[i]);
02564       gtk_grid_attach (window->grid_climbing,
02565                        GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02566       g_signal_connect (window->button_climbing[i], "toggled", window_update,
02567                         NULL);
02568     }
```

```
02569   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02570   window->spin_steps = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02572   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02573   window->label_estimates
02574     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02575   window->spin_estimates = (GtkSpinButton *)
02576     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02577   window->label_relaxation
02578     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02579   window->spin_relaxation = (GtkSpinButton *)
02580     gtk_spin_button_new_with_range (0., 2., 0.001);
02581   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
02582                    0, NCLIMBINGS, 1, 1);
02583   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
02584                    1, NCLIMBINGS, 1, 1);
02585   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
02586                    0, NCLIMBINGS + 1, 1, 1);
02587   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
02588                    1, NCLIMBINGS + 1, 1, 1);
02589   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
02590                    0, NCLIMBINGS + 2, 1, 1);
02591   gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
02592                    1, NCLIMBINGS + 2, 1, 1);
02593
02594   // Creating the array of algorithms
02595   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02596 #if !GTK4
02597   window->button_algorithm[0] = (GtkRadioButton *)
02598     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02599 #else
02600   window->button_algorithm[0] = (GtkCheckButton *)
02601     gtk_check_button_new_with_mnemonic (label_algorithm[0]);
02602 #endif
02603   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02604                                tip_algorithm[0]);
02605   gtk_grid_attach (window->grid_algorithm,
02606                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02607   g_signal_connect (window->button_algorithm[0], "toggled",
02608                     window_set_algorithm, NULL);
02609   for (i = 0; ++i < NALGORITHMS;)
02610     {
02611 #if !GTK4
02612       window->button_algorithm[i] = (GtkRadioButton *)
02613         gtk_radio_button_new_with_mnemonic
02614         (gtk_radio_button_get_group (window->button_algorithm[0]),
02615          label_algorithm[i]);
02616 #else
02617       window->button_algorithm[i] = (GtkCheckButton *)
02618         gtk_check_button_new_with_mnemonic (label_algorithm[i]);
02619       gtk_check_button_set_group (window->button_algorithm[i],
02620                                   window->button_algorithm[0]),
02621 #endif
02622         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02623                                      tip_algorithm[i]);
02624       gtk_grid_attach (window->grid_algorithm,
02625                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02626       g_signal_connect (window->button_algorithm[i], "toggled",
02627                         window_set_algorithm, NULL);
02628     }
02629   gtk_grid_attach (window->grid_algorithm,
02630                    GTK_WIDGET (window->label_simulations),
02631                    0, NALGORITHMS, 1, 1);
02632   gtk_grid_attach (window->grid_algorithm,
02633                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02634   gtk_grid_attach (window->grid_algorithm,
02635                    GTK_WIDGET (window->label_iterations),
02636                    0, NALGORITHMS + 1, 1, 1);
02637   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
02638                    1, NALGORITHMS + 1, 1, 1);
02639   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
02640                    0, NALGORITHMS + 2, 1, 1);
02641   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
02642                    1, NALGORITHMS + 2, 1, 1);
02643   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
02644                    0, NALGORITHMS + 3, 1, 1);
02645   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
02646                    1, NALGORITHMS + 3, 1, 1);
02647   gtk_grid_attach (window->grid_algorithm,
02648                    GTK_WIDGET (window->label_population),
02649                    0, NALGORITHMS + 4, 1, 1);
02650   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
02651                    1, NALGORITHMS + 4, 1, 1);
02652   gtk_grid_attach (window->grid_algorithm,
02653                    GTK_WIDGET (window->label_generations),
02654                    0, NALGORITHMS + 5, 1, 1);
02655   gtk_grid_attach (window->grid_algorithm,
```

```
02656                     GTK_WIDGET (window->spin_generations),
02657                     1, NALGORITHMS + 5, 1, 1);
02658   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
02659                     0, NALGORITHMS + 6, 1, 1);
02660   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
02661                     1, NALGORITHMS + 6, 1, 1);
02662   gtk_grid_attach (window->grid_algorithm,
02663                     GTK_WIDGET (window->label_reproduction),
02664                     0, NALGORITHMS + 7, 1, 1);
02665   gtk_grid_attach (window->grid_algorithm,
02666                     GTK_WIDGET (window->spin_reproduction),
02667                     1, NALGORITHMS + 7, 1, 1);
02668   gtk_grid_attach (window->grid_algorithm,
02669                     GTK_WIDGET (window->label_adaptation),
02670                     0, NALGORITHMS + 8, 1, 1);
02671   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),
02672                     1, NALGORITHMS + 8, 1, 1);
02673   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
02674                     0, NALGORITHMS + 9, 2, 1);
02675   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
02676                     0, NALGORITHMS + 10, 2, 1);
02677   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
02678                     0, NALGORITHMS + 11, 1, 1);
02679   gtk_grid_attach (window->grid_algorithm,
02680                     GTK_WIDGET (window->scrolled_threshold),
02681                     1, NALGORITHMS + 11, 1, 1);
02682   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02683   gtk_frame_set_child (window->frame_algorithm,
02684                        GTK_WIDGET (window->grid_algorithm));
02685
02686   // Creating the variable widgets
02687   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02688   gtk_widget_set_tooltip_text
02689     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02690   window->id_variable = g_signal_connect
02691     (window->combo_variable, "changed", window_set_variable, NULL);
02692 #if !GTK4
02693   window->button_add_variable = (GtkButton *)
02694     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02695 #else
02696   window->button_add_variable = (GtkButton *)
02697     gtk_button_new_from_icon_name ("list-add");
02698 #endif
02699   g_signal_connect (window->button_add_variable, "clicked", window_add_variable,
02700                     NULL);
02701   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02702                                _("Add variable"));
02703 #if !GTK4
02704   window->button_remove_variable = (GtkButton *)
02705     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02706 #else
02707   window->button_remove_variable = (GtkButton *)
02708     gtk_button_new_from_icon_name ("list-remove");
02709 #endif
02710   g_signal_connect (window->button_remove_variable, "clicked",
02711                     window_remove_variable, NULL);
02712   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02713                                _("Remove variable"));
02714   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02715   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02716   gtk_widget_set_tooltip_text
02717     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02718   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02719   window->id_variable_label = g_signal_connect
02720     (window->entry_variable, "changed", window_label_variable, NULL);
02721   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02722   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02723     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02724   gtk_widget_set_tooltip_text
02725     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02726   window->scrolled_min = (GtkScrolledWindow *)
02727 #if !GTK4
02728     gtk_scrolled_window_new (NULL, NULL);
02729 #else
02730     gtk_scrolled_window_new ();
02731 #endif
02732   gtk_scrolled_window_set_child (window->scrolled_min,
02733                                  GTK_WIDGET (window->spin_min));
02734   g_signal_connect (window->spin_min, "value-changed",
02735                     window_rangemin_variable, NULL);
02736   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02737   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02738     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02739   gtk_widget_set_tooltip_text
02740     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02741   window->scrolled_max = (GtkScrolledWindow *)
02742 #if !GTK4
```

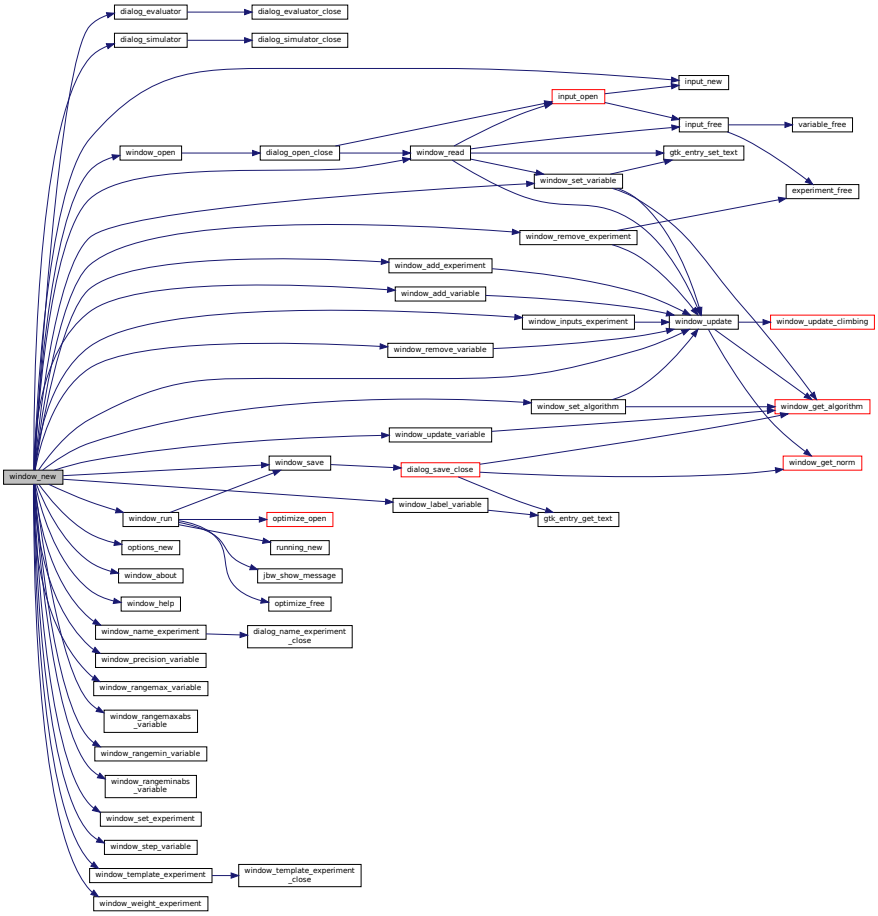
```
02743     gtk_scrolled_window_new (NULL, NULL);
02744 #else
02745     gtk_scrolled_window_new ();
02746 #endif
02747   gtk_scrolled_window_set_child (window->scrolled_max,
02748                                  GTK_WIDGET (window->spin_max));
02749   g_signal_connect (window->spin_max, "value-changed",
02750                     window_rangemax_variable, NULL);
02751   window->check_minabs = (GtkCheckButton *)
02752     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02753   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02754   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02755     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02756   gtk_widget_set_tooltip_text
02757     (GTK_WIDGET (window->spin_minabs),
02758      _("Minimum allowed value of the variable"));
02759   window->scrolled_minabs = (GtkScrolledWindow *)
02760 #if !GTK4
02761     gtk_scrolled_window_new (NULL, NULL);
02762 #else
02763     gtk_scrolled_window_new ();
02764 #endif
02765   gtk_scrolled_window_set_child (window->scrolled_minabs,
02766                                  GTK_WIDGET (window->spin_minabs));
02767   g_signal_connect (window->spin_minabs, "value-changed",
02768                     window_rangeminabs_variable, NULL);
02769   window->check_maxabs = (GtkCheckButton *)
02770     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02771   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02772   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02773     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02774   gtk_widget_set_tooltip_text
02775     (GTK_WIDGET (window->spin_maxabs),
02776      _("Maximum allowed value of the variable"));
02777   window->scrolled_maxabs = (GtkScrolledWindow *)
02778 #if !GTK4
02779     gtk_scrolled_window_new (NULL, NULL);
02780 #else
02781     gtk_scrolled_window_new ();
02782 #endif
02783   gtk_scrolled_window_set_child (window->scrolled_maxabs,
02784                                  GTK_WIDGET (window->spin_maxabs));
02785   g_signal_connect (window->spin_maxabs, "value-changed",
02786                     window_rangemaxabs_variable, NULL);
02787   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02788   window->spin_precision = (GtkSpinButton *)
02789     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02790   gtk_widget_set_tooltip_text
02791     (GTK_WIDGET (window->spin_precision),
02792      _("Number of precision floating point digits\n"
02793        "0 is for integer numbers"));
02794   g_signal_connect (window->spin_precision, "value-changed",
02795                     window_precision_variable, NULL);
02796   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02797   window->spin_sweeps =
02798     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02799   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02800                                _("Number of steps sweeping the variable"));
02801   g_signal_connect (window->spin_sweeps, "value-changed",
02802                     window_update_variable, NULL);
02803   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02804   window->spin_bits
02805     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02806   gtk_widget_set_tooltip_text
02807     (GTK_WIDGET (window->spin_bits),
02808      _("Number of bits to encode the variable"));
02809   g_signal_connect
02810     (window->spin_bits, "value-changed", window_update_variable, NULL);
02811   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02812   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02813     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02814   gtk_widget_set_tooltip_text
02815     (GTK_WIDGET (window->spin_step),
02816      _("Initial step size for the hill climbing method"));
02817   window->scrolled_step = (GtkScrolledWindow *)
02818 #if !GTK4
02819     gtk_scrolled_window_new (NULL, NULL);
02820 #else
02821     gtk_scrolled_window_new ();
02822 #endif
02823   gtk_scrolled_window_set_child (window->scrolled_step,
02824                                  GTK_WIDGET (window->spin_step));
02825   g_signal_connect
02826     (window->spin_step, "value-changed", window_step_variable, NULL);
02827   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02828   gtk_grid_attach (window->grid_variable,
02829                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
```

```
02830   gtk_grid_attach (window->grid_variable,
02831                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02832   gtk_grid_attach (window->grid_variable,
02833                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02834   gtk_grid_attach (window->grid_variable,
02835                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02836   gtk_grid_attach (window->grid_variable,
02837                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02838   gtk_grid_attach (window->grid_variable,
02839                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02840   gtk_grid_attach (window->grid_variable,
02841                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02842   gtk_grid_attach (window->grid_variable,
02843                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02844   gtk_grid_attach (window->grid_variable,
02845                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02846   gtk_grid_attach (window->grid_variable,
02847                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02848   gtk_grid_attach (window->grid_variable,
02849                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02850   gtk_grid_attach (window->grid_variable,
02851                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02852   gtk_grid_attach (window->grid_variable,
02853                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02854   gtk_grid_attach (window->grid_variable,
02855                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02856   gtk_grid_attach (window->grid_variable,
02857                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02858   gtk_grid_attach (window->grid_variable,
02859                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02860   gtk_grid_attach (window->grid_variable,
02861                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02862   gtk_grid_attach (window->grid_variable,
02863                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02864   gtk_grid_attach (window->grid_variable,
02865                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02866   gtk_grid_attach (window->grid_variable,
02867                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02868   gtk_grid_attach (window->grid_variable,
02869                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02870   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02871   gtk_frame_set_child (window->frame_variable,
02872                        GTK_WIDGET (window->grid_variable));
02873
02874   // Creating the experiment widgets
02875   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02876   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02877                                _("Experiment selector"));
02878   window->id_experiment = g_signal_connect
02879     (window->combo_experiment, "changed", window_set_experiment, NULL);
02880 #if !GTK4
02881   window->button_add_experiment = (GtkButton *)
02882     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02883 #else
02884   window->button_add_experiment = (GtkButton *)
02885     gtk_button_new_from_icon_name ("list-add");
02886 #endif
02887   g_signal_connect
02888     (window->button_add_experiment, "clicked", window_add_experiment, NULL);
02889   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02890                                _("Add experiment"));
02891 #if !GTK4
02892   window->button_remove_experiment = (GtkButton *)
02893     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02894 #else
02895   window->button_remove_experiment = (GtkButton *)
02896     gtk_button_new_from_icon_name ("list-remove");
02897 #endif
02898   g_signal_connect (window->button_remove_experiment, "clicked",
02899                     window_remove_experiment, NULL);
02900   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02901                                _("Remove experiment"));
02902   window->label_experiment
02903     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02904   window->button_experiment = (GtkButton *)
02905     gtk_button_new_with_mnemonic (_("Experimental data file"));
02906   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02907                                _("Experimental data file"));
02908   g_signal_connect (window->button_experiment, "clicked",
02909                     window_name_experiment, NULL);
02910   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02911   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02912   window->spin_weight
02913     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02914   gtk_widget_set_tooltip_text
02915     (GTK_WIDGET (window->spin_weight),
02916      _("Weight factor to build the objective function"));
```

```
02917   g_signal_connect
02918     (window->spin_weight, "value-changed", window_weight_experiment, NULL);
02919   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02920   gtk_grid_attach (window->grid_experiment,
02921                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02922   gtk_grid_attach (window->grid_experiment,
02923                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02924   gtk_grid_attach (window->grid_experiment,
02925                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02926   gtk_grid_attach (window->grid_experiment,
02927                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02928   gtk_grid_attach (window->grid_experiment,
02929                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02930   gtk_grid_attach (window->grid_experiment,
02931                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02932   gtk_grid_attach (window->grid_experiment,
02933                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02934   for (i = 0; i < MAX_NINPUTS; ++i)
02935     {
02936       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02937       window->check_template[i] = (GtkCheckButton *)
02938         gtk_check_button_new_with_label (buffer3);
02939       window->id_template[i]
02940         = g_signal_connect (window->check_template[i], "toggled",
02941                             window_inputs_experiment, NULL);
02942       gtk_grid_attach (window->grid_experiment,
02943                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02944       window->button_template[i] = (GtkButton *)
02945         gtk_button_new_with_mnemonic (_("Input template"));
02946
02947       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02948                                    _("Experimental input template file"));
02949       window->id_input[i] =
02950         g_signal_connect_swapped (window->button_template[i], "clicked",
02951                                   (GCallback) window_template_experiment,
02952                                   (void *) (size_t) i);
02953       gtk_grid_attach (window->grid_experiment,
02954                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02955     }
02956   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02957   gtk_frame_set_child (window->frame_experiment,
02958                        GTK_WIDGET (window->grid_experiment));
02959
02960   // Creating the error norm widgets
02961   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02962   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02963   gtk_frame_set_child (window->frame_norm, GTK_WIDGET (window->grid_norm));
02964 #if !GTK4
02965   window->button_norm[0] = (GtkRadioButton *)
02966     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02967 #else
02968   window->button_norm[0] = (GtkCheckButton *)
02969     gtk_check_button_new_with_mnemonic (label_norm[0]);
02970 #endif
02971   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02972                                tip_norm[0]);
02973   gtk_grid_attach (window->grid_norm,
02974                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02975   g_signal_connect (window->button_norm[0], "toggled", window_update, NULL);
02976   for (i = 0; ++i < NNORMS;)
02977     {
02978 #if !GTK4
02979       window->button_norm[i] = (GtkRadioButton *)
02980         gtk_radio_button_new_with_mnemonic
02981         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02982 #else
02983       window->button_norm[i] = (GtkCheckButton *)
02984         gtk_check_button_new_with_mnemonic (label_norm[i]);
02985       gtk_check_button_set_group (window->button_norm[i],
02986                                   window->button_norm[0]);
02987 #endif
02988       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02989                                    tip_norm[i]);
02990       gtk_grid_attach (window->grid_norm,
02991                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02992       g_signal_connect (window->button_norm[i], "toggled", window_update, NULL);
02993     }
02994   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02995   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02996   window->spin_p = (GtkSpinButton *)
02997     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02998   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02999                                _("P parameter for the P error norm"));
03000   window->scrolled_p = (GtkScrolledWindow *)
03001 #if !GTK4
03002     gtk_scrolled_window_new (NULL, NULL);
03003 #else
```

```
03004     gtk_scrolled_window_new ();
03005 #endif
03006   gtk_scrolled_window_set_child (window->scrolled_p,
03007                                  GTK_WIDGET (window->spin_p));
03008   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
03009   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
03010   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
03011                    1, 2, 1, 2);
03012
03013   // Creating the grid and attaching the widgets to the grid
03014   window->grid = (GtkGrid *) gtk_grid_new ();
03015   gtk_grid_attach (window->grid, GTK_WIDGET (window->box_buttons), 0, 0, 3, 1);
03016   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
03017   gtk_grid_attach (window->grid,
03018                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
03019   gtk_grid_attach (window->grid,
03020                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
03021   gtk_grid_attach (window->grid,
03022                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
03023   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
03024   gtk_window_set_child (window->window, GTK_WIDGET (window->grid));
03025
03026   // Setting the window logo
03027   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03028 #if !GTK4
03029   gtk_window_set_icon (window->window, window->logo);
03030 #endif
03031
03032   // Showing the window
03033 #if !GTK4
03034   gtk_widget_show_all (GTK_WIDGET (window->window));
03035 #else
03036   gtk_widget_show (GTK_WIDGET (window->window));
03037 #endif
03038
03039   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
03040 #if GTK_MINOR_VERSION >= 16
03041   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03042   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03043   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03044   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03045   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
03046   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
03047   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
03048 #endif
03049
03050   // Reading initial example
03051   input_new ();
03052   buffer2 = g_get_current_dir ();
03053   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03054   g_free (buffer2);
03055   window_read (buffer);
03056   g_free (buffer);
03057
03058 #if DEBUG_INTERFACE
03059   fprintf (stderr, "window_new:  start\n");
03060 #endif
03061 }
```

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
00038 #ifndef INTERFACE__H
00039 #define INTERFACE__H 1
00040
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00043
00048 typedef struct
00049 {
00050   GtkDialog *dialog;
00051   GtkGrid *grid;
00052   GtkLabel *label_seed;
00054   GtkSpinButton *spin_seed;
00056   GtkLabel *label_threads;
00057   GtkSpinButton *spin_threads;
00058   GtkLabel *label_climbing;
00059   GtkSpinButton *spin_climbing;
00060 } Options;
00061
00066 typedef struct
00067 {
00068   GtkDialog *dialog;
00069   GtkLabel *label;
00070   GtkSpinner *spinner;
00071   GtkGrid *grid;
00072 } Running;
00073
00078 typedef struct
00079 {
00080   GtkWindow *window;
00081   GtkGrid *grid;
00082   GtkBox *box_buttons;
00083   GtkButton *button_open;
00084   GtkButton *button_save;
00085   GtkButton *button_run;
00086   GtkButton *button_options;
00087   GtkButton *button_help;
00088   GtkButton *button_about;
00089   GtkButton *button_exit;
00090   GtkGrid *grid_files;
00091   GtkLabel *label_simulator;
00092   GtkButton *button_simulator;
00093   GtkCheckButton *check_evaluator;
00094   GtkButton *button_evaluator;
00095   GtkLabel *label_result;
00096   GtkEntry *entry_result;
00097   GtkLabel *label_variables;
00098   GtkEntry *entry_variables;
00099   GtkFrame *frame_norm;
00100   GtkGrid *grid_norm;
00101 #if !GTK4
00102   GtkRadioButton *button_norm[NNORMS];
00104 #else
00105   GtkCheckButton *button_norm[NNORMS];
00107 #endif
00108   GtkLabel *label_p;
00109   GtkSpinButton *spin_p;
00110   GtkScrolledWindow *scrolled_p;
00112   GtkFrame *frame_algorithm;
00113   GtkGrid *grid_algorithm;
00114 #if !GTK4
00115   GtkRadioButton *button_algorithm[NALGORITHMS];
```

```
00117 #else
00118   GtkCheckButton *button_algorithm[NALGORITHMS];
00120 #endif
00121   GtkLabel *label_simulations;
00122   GtkSpinButton *spin_simulations;
00124   GtkLabel *label_iterations;
00125   GtkSpinButton *spin_iterations;
00127   GtkLabel *label_tolerance;
00128   GtkSpinButton *spin_tolerance;
00129   GtkLabel *label_bests;
00130   GtkSpinButton *spin_bests;
00131   GtkLabel *label_population;
00132   GtkSpinButton *spin_population;
00134   GtkLabel *label_generations;
00135   GtkSpinButton *spin_generations;
00137   GtkLabel *label_mutation;
00138   GtkSpinButton *spin_mutation;
00139   GtkLabel *label_reproduction;
00140   GtkSpinButton *spin_reproduction;
00142   GtkLabel *label_adaptation;
00143   GtkSpinButton *spin_adaptation;
00145   GtkCheckButton *check_climbing;
00147   GtkGrid *grid_climbing;
00149 #if !GTK4
00150   GtkRadioButton *button_climbing[NCLIMBINGS];
00152 #else
00153   GtkCheckButton *button_climbing[NCLIMBINGS];
00155 #endif
00156   GtkLabel *label_steps;
00157   GtkSpinButton *spin_steps;
00158   GtkLabel *label_estimates;
00159   GtkSpinButton *spin_estimates;
00161   GtkLabel *label_relaxation;
00163   GtkSpinButton *spin_relaxation;
00165   GtkLabel *label_threshold;
00166   GtkSpinButton *spin_threshold;
00167   GtkScrolledWindow *scrolled_threshold;
00169   GtkFrame *frame_variable;
00170   GtkGrid *grid_variable;
00171   GtkComboBoxText *combo_variable;
00173   GtkButton *button_add_variable;
00174   GtkButton *button_remove_variable;
00175   GtkLabel *label_variable;
00176   GtkEntry *entry_variable;
00177   GtkLabel *label_min;
00178   GtkSpinButton *spin_min;
00179   GtkScrolledWindow *scrolled_min;
00180   GtkLabel *label_max;
00181   GtkSpinButton *spin_max;
00182   GtkScrolledWindow *scrolled_max;
00183   GtkCheckButton *check_minabs;
00184   GtkSpinButton *spin_minabs;
00185   GtkScrolledWindow *scrolled_minabs;
00186   GtkCheckButton *check_maxabs;
00187   GtkSpinButton *spin_maxabs;
00188   GtkScrolledWindow *scrolled_maxabs;
00189   GtkLabel *label_precision;
00190   GtkSpinButton *spin_precision;
00191   GtkLabel *label_sweeps;
00192   GtkSpinButton *spin_sweeps;
00193   GtkLabel *label_bits;
00194   GtkSpinButton *spin_bits;
00195   GtkLabel *label_step;
00196   GtkSpinButton *spin_step;
00197   GtkScrolledWindow *scrolled_step;
00198   GtkFrame *frame_experiment;
00199   GtkGrid *grid_experiment;
00200   GtkComboBoxText *combo_experiment;
00201   GtkButton *button_add_experiment;
00202   GtkButton *button_remove_experiment;
00203   GtkLabel *label_experiment;
00204   GtkButton *button_experiment;
00206   GtkLabel *label_weight;
00207   GtkSpinButton *spin_weight;
00208   GtkCheckButton *check_template[MAX_NINPUTS];
00210   GtkButton *button_template[MAX_NINPUTS];
00212   GdkPixbuf *logo;
00213   Experiment *experiment;
00214   Variable *variable;
00215   char *application_directory;
00216   gulong id_experiment;
00217   gulong id_experiment_name;
00218   gulong id_variable;
00219   gulong id_variable_label;
00220   gulong id_template[MAX_NINPUTS];
00222   gulong id_input[MAX_NINPUTS];
00224   unsigned int nexperiments;
```

```
00225   unsigned int nvariables;
00226 } Window;
00227
00228 // Global variables
00229 extern Window window[1];
00230
00231 // Public functions
00232 void window_new (GtkApplication * application);
00233
00234 #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 "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:

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 Function Documentation

4.15.2.1 main()

```
int main (
              int argn,
              char ** argc )
```

Main function

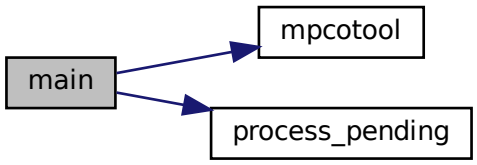
Returns

0 on succes, error code (>0) on error.

Definition at line 77 of file main.c.

```
00078 {
00079 #if HAVE_GTK
00080   show_pending = process_pending;
00081 #endif
00082   return mpcotool (argn, argc);
00083 }
```

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
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <locale.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <json-glib/json-glib.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #if HAVE_GTK
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #endif
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 #if HAVE_GTK
00067 #include "interface.h"
00068 #endif
00069 #include "mpcotool.h"
00070
00076 int
00077 main (int argn, char **argc)
00078 {
00079 #if HAVE_GTK
00080   show_pending = process_pending;
00081 #endif
00082   return mpcotool (argn, argc);
00083 }
```

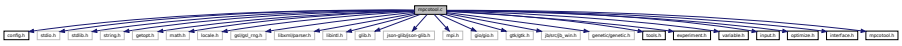
4.17 mpcotool.c File Reference

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

argn	Arguments number.
argc	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   char *buffer;
00092 #endif
00093   int o, option_index;
00094
00095   // Starting pseudo-random numbers generator
00096 #if DEBUG_MPCOTOOL
00097   fprintf (stderr, "mpcotool:  starting pseudo-random numbers generator\n");
00098 #endif
00099   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00100
00101   // Allowing spaces in the XML data file
00102 #if DEBUG_MPCOTOOL
00103   fprintf (stderr, "mpcotool:  allowing spaces in the XML data file\n");
00104 #endif
00105   xmlKeepBlanksDefault (0);
00106
00107   // Starting MPI
00108 #if HAVE_MPI
00109 #if DEBUG_MPCOTOOL
00110   fprintf (stderr, "mpcotool:  starting MPI\n");
00111 #endif
00112   MPI_Init (&argn, &argc);
00113   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00114   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00115   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00116 #else
00117   ntasks = 1;
00118 #endif
00119
00120   // Getting threads number and pseudo-random numbers generator seed
00121   nthreads_climbing = nthreads = jb_get_ncores ();
00122   optimize->seed = DEFAULT_RANDOM_SEED;
00123
00124   // Parsing command line arguments
00125   while (1)
00126     {
00127       o = getopt_long (argn, argc, "s:t:", options, &option_index);
00128       if (o == -1)
00129         break;
```

```
00130       switch (o)
00131         {
00132         case 's':
00133           optimize->seed = atol (optarg);
00134           break;
00135         case 't':
00136           nthreads_climbing = nthreads = atoi (optarg);
00137           break;
00138         default:
00139           printf ("%s\n%s\n", _("ERROR!"), _("Unknown option"));
00140           return 1;
00141         }
00142     }
00143   argn -= optind;
00144
00145   // Resetting result and variables file names
00146 #if DEBUG_MPCOTOOL
00147   fprintf (stderr, "mpcotool:  resetting result and variables file names\n");
00148 #endif
00149   input->result = input->variables = NULL;
00150
00151 #if HAVE_GTK
00152
00153   // Setting local language and international floating point numbers notation
00154   setlocale (LC_ALL, "");
00155   setlocale (LC_NUMERIC, "C");
00156   window->application_directory = g_get_current_dir ();
00157   buffer = g_build_filename (window->application_directory, LOCALE_DIR, NULL);
00158   bindtextdomain (PROGRAM_INTERFACE, buffer);
00159   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00160   textdomain (PROGRAM_INTERFACE);
00161
00162   // Initing GTK+
00163   gtk_disable_setlocale ();
00164   application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00165                                      G_APPLICATION_DEFAULT_FLAGS);
00166   g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00167
00168   // Opening the main window
00169   g_application_run (G_APPLICATION (application), 0, NULL);
00170
00171   // Freeing memory
00172   input_free ();
00173   g_free (buffer);
00174   gtk_window_destroy (window->window);
00175   g_object_unref (application);
00176   g_free (window->application_directory);
00177
00178 #else
00179
00180   // Checking syntax
00181   if (argn < 1 || argn > 3)
00182     {
00183       printf ("The syntax is:\n"
00184               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00185               "[variables_file]\n");
00186       return 2;
00187     }
00188   if (argn > 1)
00189     input->result = (char *) xmlStrdup ((xmlChar *) argc[optind + 1]);
00190   if (argn == 2)
00191     input->variables = (char *) xmlStrdup ((xmlChar *) argc[optind + 2]);
00192
00193   // Making optimization
00194 #if DEBUG_MPCOTOOL
00195   fprintf (stderr, "mpcotool:  making optimization\n");
00196 #endif
00197   if (input_open (argc[optind]))
00198     optimize_open ();
00199
00200   // Freeing memory
00201 #if DEBUG_MPCOTOOL
00202   fprintf (stderr, "mpcotool:  freeing memory and closing\n");
00203 #endif
00204   optimize_free ();
00205
00206 #endif
00207
00208   // Closing MPI
00209 #if HAVE_MPI
00210   MPI_Finalize ();
00211 #endif
00212
00213   // Freeing memory
00214   gsl_rng_free (optimize->rng);
00215
00216   // Closing
```

```
00217   return 0;
00218 }
```

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
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <getopt.h>
00044 #include <math.h>
00045 #include <locale.h>
00046 #include <gsl/gsl_rng.h>
00047 #include <libxml/parser.h>
00048 #include <libintl.h>
00049 #include <glib.h>
00050 #include <json-glib/json-glib.h>
00051 #ifdef G_OS_WIN32
00052 #include <windows.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #if HAVE_GTK
00058 #include <gio/gio.h>
00059 #include <gtk/gtk.h>
00060 #endif
00061 #include "jb/src/jb_win.h"
00062 #include "genetic/genetic.h"
00063 #include "tools.h"
00064 #include "experiment.h"
00065 #include "variable.h"
00066 #include "input.h"
00067 #include "optimize.h"
00068 #if HAVE_GTK
00069 #include "interface.h"
00070 #endif
00071 #include "mpcotool.h"
00072
00073 #define DEBUG_MPCOTOOL 1
00074
00080 int
00081 mpcotool (int argn,
00082           char **argc)
00083 {
00084   const struct option options[] = {
00085     {"seed", required_argument, NULL, 's'},
00086     {"nthreads", required_argument, NULL, 't'},
```

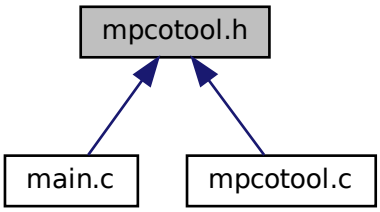
```
00087     {NULL, 0, NULL, 0}
00088   };
00089 #if HAVE_GTK
00090   GtkApplication *application;
00091   char *buffer;
00092 #endif
00093   int o, option_index;
00094
00095   // Starting pseudo-random numbers generator
00096 #if DEBUG_MPCOTOOL
00097   fprintf (stderr, "mpcotool:  starting pseudo-random numbers generator\n");
00098 #endif
00099   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00100
00101   // Allowing spaces in the XML data file
00102 #if DEBUG_MPCOTOOL
00103   fprintf (stderr, "mpcotool:  allowing spaces in the XML data file\n");
00104 #endif
00105   xmlKeepBlanksDefault (0);
00106
00107   // Starting MPI
00108 #if HAVE_MPI
00109 #if DEBUG_MPCOTOOL
00110   fprintf (stderr, "mpcotool:  starting MPI\n");
00111 #endif
00112   MPI_Init (&argn, &argc);
00113   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00114   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00115   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00116 #else
00117   ntasks = 1;
00118 #endif
00119
00120   // Getting threads number and pseudo-random numbers generator seed
00121   nthreads_climbing = nthreads = jb_get_ncores ();
00122   optimize->seed = DEFAULT_RANDOM_SEED;
00123
00124   // Parsing command line arguments
00125   while (1)
00126     {
00127       o = getopt_long (argn, argc, "s:t:", options, &option_index);
00128       if (o == -1)
00129         break;
00130       switch (o)
00131         {
00132         case 's':
00133           optimize->seed = atol (optarg);
00134           break;
00135         case 't':
00136           nthreads_climbing = nthreads = atoi (optarg);
00137           break;
00138         default:
00139           printf ("%s\n%s\n", _("ERROR!"), _("Unknown option"));
00140           return 1;
00141         }
00142     }
00143   argn -= optind;
00144
00145   // Resetting result and variables file names
00146 #if DEBUG_MPCOTOOL
00147   fprintf (stderr, "mpcotool:  resetting result and variables file names\n");
00148 #endif
00149   input->result = input->variables = NULL;
00150
00151 #if HAVE_GTK
00152
00153   // Setting local language and international floating point numbers notation
00154   setlocale (LC_ALL, "");
00155   setlocale (LC_NUMERIC, "C");
00156   window->application_directory = g_get_current_dir ();
00157   buffer = g_build_filename (window->application_directory, LOCALE_DIR, NULL);
00158   bindtextdomain (PROGRAM_INTERFACE, buffer);
00159   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00160   textdomain (PROGRAM_INTERFACE);
00161
00162   // Initing GTK+
00163   gtk_disable_setlocale ();
00164   application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00165                                      G_APPLICATION_DEFAULT_FLAGS);
00166   g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00167
00168   // Opening the main window
00169   g_application_run (G_APPLICATION (application), 0, NULL);
00170
00171   // Freeing memory
00172   input_free ();
00173   g_free (buffer);
```

```
00174   gtk_window_destroy (window->window);
00175   g_object_unref (application);
00176   g_free (window->application_directory);
00177
00178 #else
00179
00180   // Checking syntax
00181   if (argn < 1 || argn > 3)
00182     {
00183       printf ("The syntax is:\n"
00184               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00185               "[variables_file]\n");
00186       return 2;
00187     }
00188   if (argn > 1)
00189     input->result = (char *) xmlStrdup ((xmlChar *) argc[optind + 1]);
00190   if (argn == 2)
00191     input->variables = (char *) xmlStrdup ((xmlChar *) argc[optind + 2]);
00192
00193   // Making optimization
00194 #if DEBUG_MPCOTOOL
00195   fprintf (stderr, "mpcotool:  making optimization\n");
00196 #endif
00197   if (input_open (argc[optind]))
00198     optimize_open ();
00199
00200   // Freeing memory
00201 #if DEBUG_MPCOTOOL
00202   fprintf (stderr, "mpcotool:  freeing memory and closing\n");
00203 #endif
00204   optimize_free ();
00205
00206 #endif
00207
00208   // Closing MPI
00209 #if HAVE_MPI
00210   MPI_Finalize ();
00211 #endif
00212
00213   // Freeing memory
00214   gsl_rng_free (optimize->rng);
00215
00216   // Closing
00217   return 0;
00218 }
```

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

Definition in file mpcotool.h.

4.19.2 Function Documentation

4.19.2.1 mpcotool()

```
int mpcotool (
              int argn,
              char ** argc )
```

Main function.

Returns

0 on success, >0 on error.

Parameters

argn	Arguments number.
argc	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   char *buffer;
00092 #endif
00093   int o, option_index;
00094
00095   // Starting pseudo-random numbers generator
00096 #if DEBUG_MPCOTOOL
00097   fprintf (stderr, "mpcotool:  starting pseudo-random numbers generator\n");
00098 #endif
00099   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00100
00101   // Allowing spaces in the XML data file
00102 #if DEBUG_MPCOTOOL
```

```
00103   fprintf (stderr, "mpcotool:  allowing spaces in the XML data file\n");
00104 #endif
00105   xmlKeepBlanksDefault (0);
00106
00107   // Starting MPI
00108 #if HAVE_MPI
00109 #if DEBUG_MPCOTOOL
00110   fprintf (stderr, "mpcotool:  starting MPI\n");
00111 #endif
00112   MPI_Init (&argn, &argc);
00113   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00114   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00115   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00116 #else
00117   ntasks = 1;
00118 #endif
00119
00120   // Getting threads number and pseudo-random numbers generator seed
00121   nthreads_climbing = nthreads = jb_get_ncores ();
00122   optimize->seed = DEFAULT_RANDOM_SEED;
00123
00124   // Parsing command line arguments
00125   while (1)
00126     {
00127       o = getopt_long (argn, argc, "s:t:", options, &option_index);
00128       if (o == -1)
00129         break;
00130       switch (o)
00131         {
00132         case 's':
00133           optimize->seed = atol (optarg);
00134           break;
00135         case 't':
00136           nthreads_climbing = nthreads = atoi (optarg);
00137           break;
00138         default:
00139           printf ("%s\n%s\n", _("ERROR!"), _("Unknown option"));
00140           return 1;
00141         }
00142     }
00143   argn -= optind;
00144
00145   // Resetting result and variables file names
00146 #if DEBUG_MPCOTOOL
00147   fprintf (stderr, "mpcotool:  resetting result and variables file names\n");
00148 #endif
00149   input->result = input->variables = NULL;
00150
00151 #if HAVE_GTK
00152
00153   // Setting local language and international floating point numbers notation
00154   setlocale (LC_ALL, "");
00155   setlocale (LC_NUMERIC, "C");
00156   window->application_directory = g_get_current_dir ();
00157   buffer = g_build_filename (window->application_directory, LOCALE_DIR, NULL);
00158   bindtextdomain (PROGRAM_INTERFACE, buffer);
00159   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00160   textdomain (PROGRAM_INTERFACE);
00161
00162   // Initing GTK+
00163   gtk_disable_setlocale ();
00164   application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00165                                      G_APPLICATION_DEFAULT_FLAGS);
00166   g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00167
00168   // Opening the main window
00169   g_application_run (G_APPLICATION (application), 0, NULL);
00170
00171   // Freeing memory
00172   input_free ();
00173   g_free (buffer);
00174   gtk_window_destroy (window->window);
00175   g_object_unref (application);
00176   g_free (window->application_directory);
00177
00178 #else
00179
00180   // Checking syntax
00181   if (argn < 1 || argn > 3)
00182     {
00183       printf ("The syntax is:\n"
00184               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00185               "[variables_file]\n");
00186       return 2;
00187     }
00188   if (argn > 1)
00189     input->result = (char *) xmlStrdup ((xmlChar *) argc[optind + 1]);
```

```
00190   if (argn == 2)
00191     input->variables = (char *) xmlStrdup ((xmlChar *) argc[optind + 2]);
00192
00193   // Making optimization
00194 #if DEBUG_MPCOTOOL
00195   fprintf (stderr, "mpcotool:  making optimization\n");
00196 #endif
00197   if (input_open (argc[optind]))
00198     optimize_open ();
00199
00200   // Freeing memory
00201 #if DEBUG_MPCOTOOL
00202   fprintf (stderr, "mpcotool:  freeing memory and closing\n");
00203 #endif
00204   optimize_free ();
00205
00206 #endif
00207
00208   // Closing MPI
00209 #if HAVE_MPI
00210   MPI_Finalize ();
00211 #endif
00212
00213   // Freeing memory
00214   gsl_rng_free (optimize->rng);
00215
00216   // Closing
00217   return 0;
00218 }
```

4.20 mpcotool.h

Go to the documentation of this file.

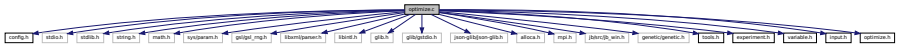
```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool.  A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and 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
00038 #ifndef MPCOTOOL__H
00039 #define MPCOTOOL__H 1
00040
00041 extern int mpcotool (int argn, char **argc);
00042
00043 #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 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 ()
- 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 ()
- 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

Definition in file optimize.c.

4.21.2 Macro Definition Documentation

4.21.2.1 DEBUG_OPTIMIZE

```
#define DEBUG_OPTIMIZE 0
```

Macro to debug optimize functions.

Definition at line 67 of file optimize.c.

4.21.2.2 RM

```
#define RM "rm"
```

Macro to define the shell remove command.

Definition at line 76 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

simulation	Simulation number.
value	Objective function value.

Definition at line 450 of file optimize.c.

```
00452 {
00453   unsigned int i, j;
00454   double e;
00455 #if DEBUG_OPTIMIZE
00456   fprintf (stderr, "optimize_best:  start\n");
00457   fprintf (stderr, "optimize_best:  nsaveds=%u nbest=%u\n",
00458            optimize->nsaveds, optimize->nbest);
00459 #endif
00460   if (optimize->nsaveds < optimize->nbest
00461       || value < optimize->error_best[optimize->nsaveds - 1])
00462     {
00463       if (optimize->nsaveds < optimize->nbest)
00464         ++optimize->nsaveds;
00465       optimize->error_best[optimize->nsaveds - 1] = value;
00466       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00467       for (i = optimize->nsaveds; --i;)
00468         {
00469           if (optimize->error_best[i] < optimize->error_best[i - 1])
00470             {
00471               j = optimize->simulation_best[i];
00472               e = optimize->error_best[i];
```

```
00473               optimize->simulation_best[i] = optimize->simulation_best[i - 1];
00474               optimize->error_best[i] = optimize->error_best[i - 1];
00475               optimize->simulation_best[i - 1] = j;
00476               optimize->error_best[i - 1] = e;
00477             }
00478           else
00479             break;
00480         }
00481     }
00482 #if DEBUG_OPTIMIZE
00483   fprintf (stderr, "optimize_best:  end\n");
00484 #endif
00485 }
```

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

simulation	Simulation number.
value	Objective function value.

Definition at line 812 of file optimize.c.

```
00814 {
00815 #if DEBUG_OPTIMIZE
00816   fprintf (stderr, "optimize_best_climbing:  start\n");
00817   fprintf (stderr,
00818            "optimize_best_climbing:  simulation=%u value=%.14le best=%.14le\n",
00819            simulation, value, optimize->error_best[0]);
00820 #endif
00821   if (value < optimize->error_best[0])
00822     {
00823       optimize->error_best[0] = value;
00824       optimize->simulation_best[0] = simulation;
00825 #if DEBUG_OPTIMIZE
00826       fprintf (stderr,
00827                "optimize_best_climbing:  BEST simulation=%u value=%.14le\n",
00828                simulation, value);
00829 #endif
00830     }
00831 #if DEBUG_OPTIMIZE
00832   fprintf (stderr, "optimize_best_climbing:  end\n");
00833 #endif
00834 }
```

4.21.3.3 optimize_climbing()

```
static void optimize_climbing ( )  [inline], [static]
```

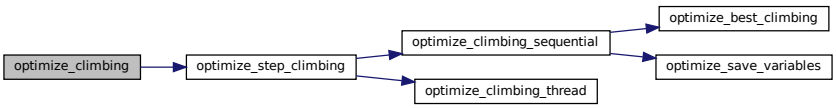
Function to optimize with a hill climbing method.

Definition at line 1040 of file optimize.c.

```
01041 {
01042   unsigned int i, j, k, b, s, adjust;
01043 #if DEBUG_OPTIMIZE
01044   fprintf (stderr, "optimize_climbing:  start\n");
01045 #endif
```

```
01046   for (i = 0; i < optimize->nvariables; ++i)
01047     optimize->climbing[i] = 0.;
01048   b = optimize->simulation_best[0] * optimize->nvariables;
01049   s = optimize->nsimulations;
01050   adjust = 1;
01051   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01052     {
01053 #if DEBUG_OPTIMIZE
01054       fprintf (stderr, "optimize_climbing:  step=%u old_best=%u\n",
01055                i, optimize->simulation_best[0]);
01056 #endif
01057       optimize_step_climbing (s);
01058       k = optimize->simulation_best[0] * optimize->nvariables;
01059 #if DEBUG_OPTIMIZE
01060       fprintf (stderr, "optimize_climbing:  step=%u best=%u\n",
01061                i, optimize->simulation_best[0]);
01062 #endif
01063       if (k == b)
01064         {
01065           if (adjust)
01066             for (j = 0; j < optimize->nvariables; ++j)
01067               optimize->step[j] *= 0.5;
01068           for (j = 0; j < optimize->nvariables; ++j)
01069             optimize->climbing[j] = 0.;
01070           adjust = 1;
01071         }
01072       else
01073         {
01074           for (j = 0; j < optimize->nvariables; ++j)
01075             {
01076 #if DEBUG_OPTIMIZE
01077               fprintf (stderr,
01078                        "optimize_climbing:  best%u=%.14le old%u=%.14le\n",
01079                        j, optimize->value[k + j], j, optimize->value[b + j]);
01080 #endif
01081               optimize->climbing[j]
01082                 = (1.  - optimize->relaxation) * optimize->climbing[j]
01083                 + optimize->relaxation
01084                 * (optimize->value[k + j] - optimize->value[b + j]);
01085 #if DEBUG_OPTIMIZE
01086               fprintf (stderr, "optimize_climbing:  climbing%u=%.14le\n",
01087                        j, optimize->climbing[j]);
01088 #endif
01089             }
01090           adjust = 0;
01091         }
01092     }
01093 #if DEBUG_OPTIMIZE
01094   fprintf (stderr, "optimize_climbing:  end\n");
01095 #endif
01096 }
```

Here is the call graph for this function:



4.21.3.4 optimize_climbing_sequential()

```
static void optimize_climbing_sequential (
             unsigned int simulation )  [inline], [static]
```

Function to estimate the hill climbing sequentially.

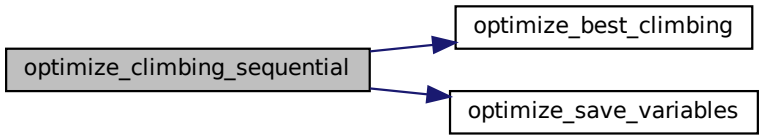
Parameters

simulation	Simulation number.

Definition at line 840 of file optimize.c.

```
00841 {
00842   double e;
00843   unsigned int i, j;
00844 #if DEBUG_OPTIMIZE
00845   fprintf (stderr, "optimize_climbing_sequential:  start\n");
00846   fprintf (stderr, "optimize_climbing_sequential:  nstart_climbing=%u "
00847            "nend_climbing=%u\n",
00848            optimize->nstart_climbing, optimize->nend_climbing);
00849 #endif
00850   for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00851     {
00852       j = simulation + i;
00853       e = optimize_norm (j);
00854       optimize_best_climbing (j, e);
00855       optimize_save_variables (j, e);
00856       if (e < optimize->threshold)
00857         {
00858           optimize->stop = 1;
00859           break;
00860         }
00861 #if DEBUG_OPTIMIZE
00862       fprintf (stderr, "optimize_climbing_sequential:  i=%u e=%lg\n", i, e);
00863 #endif
00864     }
00865 #if DEBUG_OPTIMIZE
00866   fprintf (stderr, "optimize_climbing_sequential:  end\n");
00867 #endif
00868 }
```

Here is the call graph for this function:



4.21.3.5 optimize_climbing_thread()

```
static void * optimize_climbing_thread (
             ParallelData * data )  [static]
```

Function to estimate the hill climbing on a thread.

Returns

NULL

Parameters

data	Function data.

Definition at line 876 of file optimize.c.

```
00877 {
00878   unsigned int i, thread;
00879   double e;
00880 #if DEBUG_OPTIMIZE
00881   fprintf (stderr, "optimize_climbing_thread:  start\n");
00882 #endif
00883   thread = data->thread;
00884 #if DEBUG_OPTIMIZE
00885   fprintf (stderr, "optimize_climbing_thread:  thread=%u start=%u end=%u\n",
00886            thread,
00887            optimize->thread_climbing[thread],
00888            optimize->thread_climbing[thread + 1]);
00889 #endif
00890   for (i = optimize->thread_climbing[thread];
00891        i < optimize->thread_climbing[thread + 1]; ++i)
00892     {
00893       e = optimize_norm (i);
00894       g_mutex_lock (mutex);
00895       optimize_best_climbing (i, e);
00896       optimize_save_variables (i, e);
00897       if (e < optimize->threshold)
00898         optimize->stop = 1;
00899       g_mutex_unlock (mutex);
00900       if (optimize->stop)
00901         break;
00902 #if DEBUG_OPTIMIZE
00903       fprintf (stderr, "optimize_climbing_thread:  i=%u e=%lg\n", i, e);
00904 #endif
00905     }
00906 #if DEBUG_OPTIMIZE
00907   fprintf (stderr, "optimize_climbing_thread:  end\n");
00908 #endif
00909   g_thread_exit (NULL);
00910   return NULL;
00911 }
```

4.21.3.6 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

variable	Variable number.
estimate	Estimate number.

Definition at line 941 of file optimize.c.

```
00945 {
00946   double x;
00947 #if DEBUG_OPTIMIZE
00948   fprintf (stderr, "optimize_estimate_climbing_coordinates:  start\n");
00949 #endif
00950   x = optimize->climbing[variable];
00951   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00952     {
00953       if (estimate & 1)
00954         x += optimize->step[variable];
00955       else
00956         x -= optimize->step[variable];
00957     }
```

```
00958 #if DEBUG_OPTIMIZE
00959   fprintf (stderr,
00960            "optimize_estimate_climbing_coordinates:  climbing%u=%lg\n",
00961            variable, x);
00962   fprintf (stderr, "optimize_estimate_climbing_coordinates:  end\n");
00963 #endif
00964   return x;
00965 }
```

4.21.3.7 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

variable	Variable number.
estimate	Estimate number.

Definition at line 917 of file optimize.c.

```
00922 {
00923   double x;
00924 #if DEBUG_OPTIMIZE
00925   fprintf (stderr, "optimize_estimate_climbing_random:  start\n");
00926 #endif
00927   x = optimize->climbing[variable]
00928     + (1.  - 2.  * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00929 #if DEBUG_OPTIMIZE
00930   fprintf (stderr, "optimize_estimate_climbing_random:  climbing%u=%lg\n",
00931            variable, x);
00932   fprintf (stderr, "optimize_estimate_climbing_random:  end\n");
00933 #endif
00934   return x;
00935 }
```

4.21.3.8 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the Optimize struct.

Definition at line 1397 of file optimize.c.

```
01398 {
01399   unsigned int i, j;
01400 #if DEBUG_OPTIMIZE
01401   fprintf (stderr, "optimize_free:  start\n");
01402 #endif
01403   for (j = 0; j < optimize->ninputs; ++j)
01404     {
01405       for (i = 0; i < optimize->nexperiments; ++i)
01406         g_mapped_file_unref (optimize->file[j][i]);
01407       g_free (optimize->file[j]);
01408     }
01409   g_free (optimize->error_old);
01410   g_free (optimize->value_old);
01411   g_free (optimize->value);
01412   g_free (optimize->genetic_variable);
01413 #if DEBUG_OPTIMIZE
01414   fprintf (stderr, "optimize_free:  end\n");
01415 #endif
01416 }
```

4.21.3.9 optimize_genetic()

```
static void optimize_genetic ( )  [static]
```

Function to optimize with the genetic algorithm.

Definition at line 1137 of file optimize.c.

```
01138 {
01139   double *best_variable = NULL;
01140   char *best_genome = NULL;
01141   double best_objective = 0.;
01142 #if DEBUG_OPTIMIZE
01143   fprintf (stderr, "optimize_genetic:  start\n");
01144   fprintf (stderr, "optimize_genetic:  ntasks=%u nthreads=%u\n", ntasks,
01145            nthreads);
01146   fprintf (stderr,
01147            "optimize_genetic:  nvariables=%u population=%u generations=%u\n",
01148            optimize->nvariables, optimize->nsimulations, optimize->niterations);
01149   fprintf (stderr,
01150            "optimize_genetic:  mutation=%lg reproduction=%lg adaptation=%lg\n",
01151            optimize->mutation_ratio, optimize->reproduction_ratio,
01152            optimize->adaptation_ratio);
01153 #endif
01154   genetic_algorithm_default (optimize->nvariables,
01155                              optimize->genetic_variable,
01156                              optimize->nsimulations,
01157                              optimize->niterations,
01158                              optimize->mutation_ratio,
01159                              optimize->reproduction_ratio,
01160                              optimize->adaptation_ratio,
01161                              optimize->seed,
01162                              optimize->threshold,
01163                              &optimize_genetic_objective,
01164                              &best_genome, &best_variable, &best_objective);
01165 #if DEBUG_OPTIMIZE
01166   fprintf (stderr, "optimize_genetic:  the best\n");
01167 #endif
01168   optimize->error_old = (double *) g_malloc (sizeof (double));
01169   optimize->value_old
01170     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01171   optimize->error_old[0] = best_objective;
01172   memcpy (optimize->value_old, best_variable,
01173           optimize->nvariables * sizeof (double));
01174   g_free (best_genome);
01175   g_free (best_variable);
01176   optimize_print ();
01177 #if DEBUG_OPTIMIZE
01178   fprintf (stderr, "optimize_genetic:  end\n");
01179 #endif
01180 }
```

4.21.3.10 optimize_genetic_objective()

```
static double optimize_genetic_objective (
               Entity * entity )  [static]
```

Function to calculate the objective function of an entity.

Returns

objective function value.

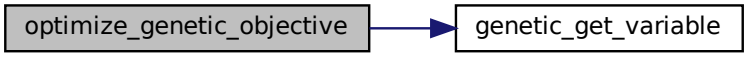
Parameters

entity	entity data.

Definition at line 1104 of file optimize.c.

```
01105 {
01106   unsigned int j;
01107   double objective;
01108   char buffer[64];
01109 #if DEBUG_OPTIMIZE
01110   fprintf (stderr, "optimize_genetic_objective:  start\n");
01111 #endif
01112   for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114       optimize->value[entity->id * optimize->nvariables + j]
01115         = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117   objective = optimize_norm (entity->id);
01118   g_mutex_lock (mutex);
01119   for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122       fprintf (optimize->file_variables, buffer,
01123                genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125   fprintf (optimize->file_variables, "%.14le\n", objective);
01126   g_mutex_unlock (mutex);
01127 #if DEBUG_OPTIMIZE
01128   fprintf (stderr, "optimize_genetic_objective:  end\n");
01129 #endif
01130   return objective;
01131 }
```

Here is the call graph for this function:



4.21.3.11 optimize_input()

```
static void optimize_input (
             unsigned int simulation,
             char * input,
             GMappedFile * stencil )  [inline], [static]
```

Function to write the simulation input file.

Parameters

simulation	Simulation number.
input	Input file name.
stencil	Template of the input file name.

Definition at line 95 of file optimize.c.

```
00098 {
00099   char buffer[32], value[32];
00100   GRegex *regex;
00101   FILE *file;
00102   char *buffer2, *buffer3 = NULL, *content;
00103   gsize length;
```

```
00104   unsigned int i;
00105
00106 #if DEBUG_OPTIMIZE
00107   fprintf (stderr, "optimize_input:  start\n");
00108 #endif
00109
00110   // Checking the file
00111   if (!stencil)
00112     goto optimize_input_end;
00113
00114   // Opening stencil
00115   content = g_mapped_file_get_contents (stencil);
00116   length = g_mapped_file_get_length (stencil);
00117 #if DEBUG_OPTIMIZE
00118   fprintf (stderr, "optimize_input:  length=%lu\ncontent:\n%s", length, content);
00119 #endif
00120   file = g_fopen (input, "w");
00121
00122   // Parsing stencil
00123   for (i = 0; i < optimize->nvariables; ++i)
00124     {
00125 #if DEBUG_OPTIMIZE
00126       fprintf (stderr, "optimize_input:  variable=%u\n", i);
00127 #endif
00128       snprintf (buffer, 32, "@variable%u@", i + 1);
00129       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00130                            NULL);
00131       if (i == 0)
00132         {
00133           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00134                                              optimize->label[i],
00135                                              (GRegexMatchFlags) 0, NULL);
00136 #if DEBUG_OPTIMIZE
00137           fprintf (stderr, "optimize_input:  buffer2\n%s", buffer2);
00138 #endif
00139         }
00140       else
00141         {
00142           length = strlen (buffer3);
00143           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00144                                              optimize->label[i],
00145                                              (GRegexMatchFlags) 0, NULL);
00146           g_free (buffer3);
00147         }
00148       g_regex_unref (regex);
00149       length = strlen (buffer2);
00150       snprintf (buffer, 32, "@value%u@", i + 1);
00151       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00152                            NULL);
00153       snprintf (value, 32, format[optimize->precision[i]],
00154                 optimize->value[simulation * optimize->nvariables + i]);
00155
00156 #if DEBUG_OPTIMIZE
00157       fprintf (stderr, "optimize_input:  value=%s\n", value);
00158 #endif
00159       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00160                                          (GRegexMatchFlags) 0, NULL);
00161       g_free (buffer2);
00162       g_regex_unref (regex);
00163     }
00164
00165   // Saving input file
00166   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00167   g_free (buffer3);
00168   fclose (file);
00169
00170 optimize_input_end:
00171 #if DEBUG_OPTIMIZE
00172   fprintf (stderr, "optimize_input:  end\n");
00173 #endif
00174   return;
00175 }
```

4.21.3.12 optimize_iterate()

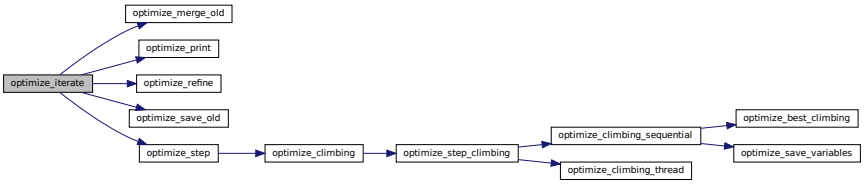
```
static void optimize_iterate ( )  [inline], [static]
```

Function to iterate the algorithm.

Definition at line 1367 of file optimize.c.

```
01368 {
01369   unsigned int i;
01370 #if DEBUG_OPTIMIZE
01371   fprintf (stderr, "optimize_iterate:  start\n");
01372 #endif
01373   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01374   optimize->value_old =
01375     (double *) g_malloc (optimize->nbest * optimize->nvariables *
01376                          sizeof (double));
01377   optimize_step ();
01378   optimize_save_old ();
01379   optimize_refine ();
01380   optimize_print ();
01381   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01382     {
01383       optimize_step ();
01384       optimize_merge_old ();
01385       optimize_refine ();
01386       optimize_print ();
01387     }
01388 #if DEBUG_OPTIMIZE
01389   fprintf (stderr, "optimize_iterate:  end\n");
01390 #endif
01391 }
```

Here is the call graph for this function:



4.21.3.13 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

nsaveds	Number of saved results.
simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 563 of file optimize.c.

```
00568 {
00569   unsigned int i, j, k, s[optimize->nbest];
00570   double e[optimize->nbest];
00571 #if DEBUG_OPTIMIZE
00572   fprintf (stderr, "optimize_merge:  start\n");
00573 #endif
00574   i = j = k = 0;
00575   do
00576     {
```

```
00577       if (i == optimize->nsaveds)
00578         {
00579           s[k] = simulation_best[j];
00580           e[k] = error_best[j];
00581           ++j;
00582           ++k;
00583           if (j == nsaveds)
00584             break;
00585         }
00586       else if (j == nsaveds)
00587         {
00588           s[k] = optimize->simulation_best[i];
00589           e[k] = optimize->error_best[i];
00590           ++i;
00591           ++k;
00592           if (i == optimize->nsaveds)
00593             break;
00594         }
00595       else if (optimize->error_best[i] > error_best[j])
00596         {
00597           s[k] = simulation_best[j];
00598           e[k] = error_best[j];
00599           ++j;
00600           ++k;
00601         }
00602       else
00603         {
00604           s[k] = optimize->simulation_best[i];
00605           e[k] = optimize->error_best[i];
00606           ++i;
00607           ++k;
00608         }
00609     }
00610   while (k < optimize->nbest);
00611   optimize->nsaveds = k;
00612   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00613   memcpy (optimize->error_best, e, k * sizeof (double));
00614 #if DEBUG_OPTIMIZE
00615   fprintf (stderr, "optimize_merge:  end\n");
00616 #endif
00617 }
```

4.21.3.14 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 1218 of file optimize.c.

```
01219 {
01220   unsigned int i, j, k;
01221   double v[optimize->nbest * optimize->nvariables], e[optimize->nbest],
01222     *enew, *eold;
01223 #if DEBUG_OPTIMIZE
01224   fprintf (stderr, "optimize_merge_old:  start\n");
01225 #endif
01226   enew = optimize->error_best;
01227   eold = optimize->error_old;
01228   i = j = k = 0;
01229   do
01230     {
01231       if (*enew < *eold)
01232         {
01233           memcpy (v + k * optimize->nvariables,
01234                   optimize->value
01235                   + optimize->simulation_best[i] * optimize->nvariables,
01236                   optimize->nvariables * sizeof (double));
01237           e[k] = *enew;
01238           ++k;
01239           ++enew;
01240           ++i;
01241         }
01242       else
01243         {
01244           memcpy (v + k * optimize->nvariables,
01245                   optimize->value_old + j * optimize->nvariables,
01246                   optimize->nvariables * sizeof (double));
01247           e[k] = *eold;
```

```
01248           ++k;
01249           ++eold;
01250           ++j;
01251         }
01252     }
01253   while (k < optimize->nbest);
01254   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01255   memcpy (optimize->error_old, e, k * sizeof (double));
01256 #if DEBUG_OPTIMIZE
01257   fprintf (stderr, "optimize_merge_old:  end\n");
01258 #endif
01259 }
```

4.21.3.15 optimize_MonteCarlo()

```
static void optimize_MonteCarlo ( )  [static]
```

Function to optimize with the Monte-Carlo algorithm.

Definition at line 721 of file optimize.c.

```
00722 {
00723   unsigned int i, j;
00724   GThread *thread[nthreads];
00725   ParallelData data[nthreads];
00726 #if DEBUG_OPTIMIZE
00727   fprintf (stderr, "optimize_MonteCarlo:  start\n");
00728 #endif
00729   for (i = 0; i < optimize->nsimulations; ++i)
00730     for (j = 0; j < optimize->nvariables; ++j)
00731       optimize->value[i * optimize->nvariables + j]
00732         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00733         * (optimize->rangemax[j] - optimize->rangemin[j]);
00734   optimize->nsaveds = 0;
00735   if (nthreads <= 1)
00736     optimize_sequential ();
00737   else
00738     {
00739       for (i = 0; i < nthreads; ++i)
00740         {
00741           data[i].thread = i;
00742           thread[i]
00743             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00744         }
00745       for (i = 0; i < nthreads; ++i)
00746         g_thread_join (thread[i]);
00747     }
00748 #if HAVE_MPI
00749   // Communicating tasks results
00750   optimize_synchronise ();
00751 #endif
00752 #if DEBUG_OPTIMIZE
00753   fprintf (stderr, "optimize_MonteCarlo:  end\n");
00754 #endif
00755 }
```

4.21.3.16 optimize_norm_euclidian()

```
static double optimize_norm_euclidian (
             unsigned int simulation )  [static]
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

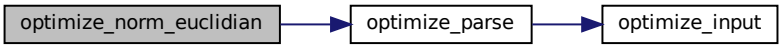
Parameters

simulation	simulation number.

Definition at line 298 of file optimize.c.

```
00299 {
00300   double e, ei;
00301   unsigned int i;
00302 #if DEBUG_OPTIMIZE
00303   fprintf (stderr, "optimize_norm_euclidian:  start\n");
00304 #endif
00305   e = 0.;
00306   for (i = 0; i < optimize->nexperiments; ++i)
00307     {
00308       ei = optimize_parse (simulation, i);
00309       e += ei * ei;
00310     }
00311   e = sqrt (e);
00312 #if DEBUG_OPTIMIZE
00313   fprintf (stderr, "optimize_norm_euclidian:  error=%lg\n", e);
00314   fprintf (stderr, "optimize_norm_euclidian:  end\n");
00315 #endif
00316   return e;
00317 }
```

Here is the call graph for this function:



4.21.3.17 optimize_norm_maximum()

```
static double optimize_norm_maximum (
              unsigned int simulation )  [static]
```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

Parameters

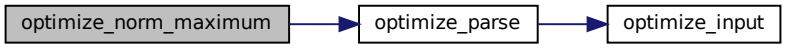
simulation	simulation number.

Definition at line 325 of file optimize.c.

```
00326 {
00327   double e, ei;
00328   unsigned int i;
00329 #if DEBUG_OPTIMIZE
00330   fprintf (stderr, "optimize_norm_maximum:  start\n");
00331 #endif
00332   e = 0.;
00333   for (i = 0; i < optimize->nexperiments; ++i)
```

```
00334     {
00335       ei = fabs (optimize_parse (simulation, i));
00336       e = fmax (e, ei);
00337     }
00338 #if DEBUG_OPTIMIZE
00339   fprintf (stderr, "optimize_norm_maximum:  error=%lg\n", e);
00340   fprintf (stderr, "optimize_norm_maximum:  end\n");
00341 #endif
00342   return e;
00343 }
```

Here is the call graph for this function:



4.21.3.18 optimize_norm_p()

```
static double optimize_norm_p (
              unsigned int simulation )  [static]
```

Function to calculate the P error norm.

Returns

P error norm.

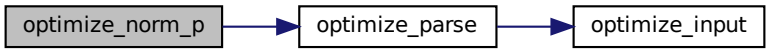
Parameters

simulation	simulation number.

Definition at line 351 of file optimize.c.

```
00352 {
00353   double e, ei;
00354   unsigned int i;
00355 #if DEBUG_OPTIMIZE
00356   fprintf (stderr, "optimize_norm_p:  start\n");
00357 #endif
00358   e = 0.;
00359   for (i = 0; i < optimize->nexperiments; ++i)
00360     {
00361       ei = fabs (optimize_parse (simulation, i));
00362       e += pow (ei, optimize->p);
00363     }
00364   e = pow (e, 1.  / optimize->p);
00365 #if DEBUG_OPTIMIZE
00366   fprintf (stderr, "optimize_norm_p:  error=%lg\n", e);
00367   fprintf (stderr, "optimize_norm_p:  end\n");
00368 #endif
00369   return e;
00370 }
```

Here is the call graph for this function:



4.21.3.19 optimize_norm_taxicab()

```
static double optimize_norm_taxicab (
             unsigned int simulation )  [static]
```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

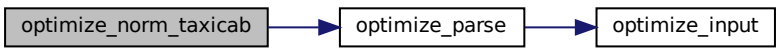
Parameters

simulation	simulation number.

Definition at line 378 of file optimize.c.

```
00379 {
00380   double e;
00381   unsigned int i;
00382 #if DEBUG_OPTIMIZE
00383   fprintf (stderr, "optimize_norm_taxicab:  start\n");
00384 #endif
00385   e = 0.;
00386   for (i = 0; i < optimize->nexperiments; ++i)
00387     e += fabs (optimize_parse (simulation, i));
00388 #if DEBUG_OPTIMIZE
00389   fprintf (stderr, "optimize_norm_taxicab:  error=%lg\n", e);
00390   fprintf (stderr, "optimize_norm_taxicab:  end\n");
00391 #endif
00392   return e;
00393 }
```

Here is the call graph for this function:



4.21.3.20 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1422 of file optimize.c.

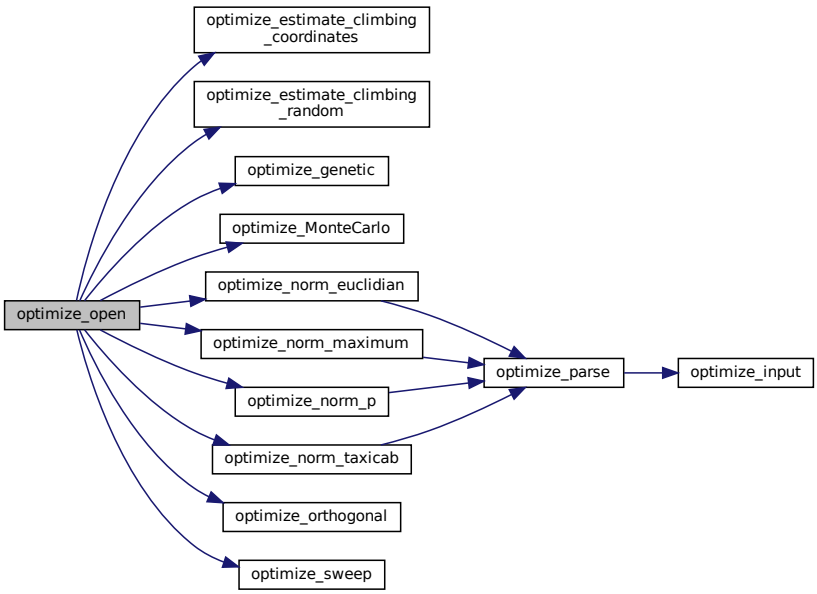
```
01423 {
01424   GTimeZone *tz;
01425   GDateTime *t0, *t;
01426   unsigned int i, j;
01427
01428 #if DEBUG_OPTIMIZE
01429   char *buffer;
01430   fprintf (stderr, "optimize_open:  start\n");
01431 #endif
01432
01433   // Getting initial time
01434 #if DEBUG_OPTIMIZE
01435   fprintf (stderr, "optimize_open:  getting initial time\n");
01436 #endif
01437   tz = g_time_zone_new_utc ();
01438   t0 = g_date_time_new_now (tz);
01439
01440   // Obtaining and initing the pseudo-random numbers generator seed
01441 #if DEBUG_OPTIMIZE
01442   fprintf (stderr, "optimize_open:  getting initial seed\n");
01443 #endif
01444   if (optimize->seed == DEFAULT_RANDOM_SEED)
01445     optimize->seed = input->seed;
01446   gsl_rng_set (optimize->rng, optimize->seed);
01447
01448   // Replacing the working directory
01449 #if DEBUG_OPTIMIZE
01450   fprintf (stderr, "optimize_open:  replacing the working directory\n");
01451 #endif
01452   g_chdir (input->directory);
01453
01454   // Getting results file names
01455   optimize->result = input->result;
01456   optimize->variables = input->variables;
01457
01458   // Obtaining the simulator file
01459   optimize->simulator = input->simulator;
01460
01461   // Obtaining the evaluator file
01462   optimize->evaluator = input->evaluator;
01463
01464   // Reading the algorithm
01465   optimize->algorithm = input->algorithm;
01466   switch (optimize->algorithm)
01467     {
01468     case ALGORITHM_MONTE_CARLO:
01469       optimize_algorithm = optimize_MonteCarlo;
01470       break;
01471     case ALGORITHM_SWEEP:
01472       optimize_algorithm = optimize_sweep;
01473       break;
01474     case ALGORITHM_ORTHOGONAL:
01475       optimize_algorithm = optimize_orthogonal;
01476       break;
01477     default:
01478       optimize_algorithm = optimize_genetic;
01479       optimize->mutation_ratio = input->mutation_ratio;
01480       optimize->reproduction_ratio = input->reproduction_ratio;
01481       optimize->adaptation_ratio = input->adaptation_ratio;
01482     }
01483   optimize->nvariables = input->nvariables;
01484   optimize->nsimulations = input->nsimulations;
01485   optimize->niterations = input->niterations;
01486   optimize->nbest = input->nbest;
01487   optimize->tolerance = input->tolerance;
01488   optimize->nsteps = input->nsteps;
01489   optimize->nestimates = 0;
01490   optimize->threshold = input->threshold;
01491   optimize->stop = 0;
01492   if (input->nsteps)
01493     {
01494       optimize->relaxation = input->relaxation;
01495       switch (input->climbing)
01496         {
01497         case CLIMBING_METHOD_COORDINATES:
01498           optimize->nestimates = 2 * optimize->nvariables;
```

```
01499           optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
01500           break;
01501         default:
01502           optimize->nestimates = input->nestimates;
01503           optimize_estimate_climbing = optimize_estimate_climbing_random;
01504         }
01505     }
01506
01507 #if DEBUG_OPTIMIZE
01508   fprintf (stderr, "optimize_open:  nbest=%u\n", optimize->nbest);
01509 #endif
01510   optimize->simulation_best
01511     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01512   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01513
01514   // Reading the experimental data
01515 #if DEBUG_OPTIMIZE
01516   buffer = g_get_current_dir ();
01517   fprintf (stderr, "optimize_open:  current directory=%s\n", buffer);
01518   g_free (buffer);
01519 #endif
01520   optimize->nexperiments = input->nexperiments;
01521   optimize->ninputs = input->experiment->ninputs;
01522   optimize->experiment
01523     = (char **) alloca (input->nexperiments * sizeof (char *));
01524   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01525   for (i = 0; i < input->experiment->ninputs; ++i)
01526     optimize->file[i] = (GMappedFile **)
01527       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01528   for (i = 0; i < input->nexperiments; ++i)
01529     {
01530 #if DEBUG_OPTIMIZE
01531       fprintf (stderr, "optimize_open:  i=%u\n", i);
01532 #endif
01533       optimize->experiment[i] = input->experiment[i].name;
01534       optimize->weight[i] = input->experiment[i].weight;
01535 #if DEBUG_OPTIMIZE
01536       fprintf (stderr, "optimize_open:  experiment=%s weight=%lg\n",
01537                optimize->experiment[i], optimize->weight[i]);
01538 #endif
01539       for (j = 0; j < input->experiment->ninputs; ++j)
01540         {
01541 #if DEBUG_OPTIMIZE
01542           fprintf (stderr, "optimize_open:  stencil%u\n", j + 1);
01543 #endif
01544           optimize->file[j][i]
01545             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01546         }
01547     }
01548
01549   // Reading the variables data
01550 #if DEBUG_OPTIMIZE
01551   fprintf (stderr, "optimize_open:  reading variables\n");
01552 #endif
01553   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01554   j = input->nvariables * sizeof (double);
01555   optimize->rangemin = (double *) alloca (j);
01556   optimize->rangeminabs = (double *) alloca (j);
01557   optimize->rangemax = (double *) alloca (j);
01558   optimize->rangemaxabs = (double *) alloca (j);
01559   optimize->step = (double *) alloca (j);
01560   j = input->nvariables * sizeof (unsigned int);
01561   optimize->precision = (unsigned int *) alloca (j);
01562   optimize->nsweeps = (unsigned int *) alloca (j);
01563   optimize->nbits = (unsigned int *) alloca (j);
01564   for (i = 0; i < input->nvariables; ++i)
01565     {
01566       optimize->label[i] = input->variable[i].name;
01567       optimize->rangemin[i] = input->variable[i].rangemin;
01568       optimize->rangeminabs[i] = input->variable[i].rangeminabs;
01569       optimize->rangemax[i] = input->variable[i].rangemax;
01570       optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
01571       optimize->precision[i] = input->variable[i].precision;
01572       optimize->step[i] = input->variable[i].step;
01573       optimize->nsweeps[i] = input->variable[i].nsweeps;
01574       optimize->nbits[i] = input->variable[i].nbits;
01575     }
01576   if (input->algorithm == ALGORITHM_SWEEP
01577       || input->algorithm == ALGORITHM_ORTHOGONAL)
01578     {
01579       optimize->nsimulations = 1;
01580       for (i = 0; i < input->nvariables; ++i)
01581         {
01582           optimize->nsimulations *= optimize->nsweeps[i];
01583 #if DEBUG_OPTIMIZE
01584           fprintf (stderr, "optimize_open:  nsweeps=%u nsimulations=%u\n",
01585                    optimize->nsweeps[i], optimize->nsimulations);
```

```
01586 #endif
01587         }
01588     }
01589   if (optimize->nsteps)
01590     optimize->climbing
01591       = (double *) alloca (optimize->nvariables * sizeof (double));
01592
01593   // Setting error norm
01594   switch (input->norm)
01595     {
01596     case ERROR_NORM_EUCLIDIAN:
01597       optimize_norm = optimize_norm_euclidian;
01598       break;
01599     case ERROR_NORM_MAXIMUM:
01600       optimize_norm = optimize_norm_maximum;
01601       break;
01602     case ERROR_NORM_P:
01603       optimize_norm = optimize_norm_p;
01604       optimize->p = input->p;
01605       break;
01606     default:
01607       optimize_norm = optimize_norm_taxicab;
01608     }
01609
01610   // Allocating values
01611 #if DEBUG_OPTIMIZE
01612   fprintf (stderr, "optimize_open:  allocating variables\n");
01613   fprintf (stderr, "optimize_open:  nvariables=%u algorithm=%u\n",
01614            optimize->nvariables, optimize->algorithm);
01615 #endif
01616   optimize->genetic_variable = NULL;
01617   if (optimize->algorithm == ALGORITHM_GENETIC)
01618     {
01619       optimize->genetic_variable = (GeneticVariable *)
01620         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01621       for (i = 0; i < optimize->nvariables; ++i)
01622         {
01623 #if DEBUG_OPTIMIZE
01624           fprintf (stderr, "optimize_open:  i=%u min=%lg max=%lg nbits=%u\n",
01625                    i, optimize->rangemin[i], optimize->rangemax[i],
01626                    optimize->nbits[i]);
01627 #endif
01628           optimize->genetic_variable[i].minimum = optimize->rangemin[i];
01629           optimize->genetic_variable[i].maximum = optimize->rangemax[i];
01630           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01631         }
01632     }
01633 #if DEBUG_OPTIMIZE
01634   fprintf (stderr, "optimize_open:  nvariables=%u nsimulations=%u\n",
01635            optimize->nvariables, optimize->nsimulations);
01636 #endif
01637   optimize->value = (double *)
01638     g_malloc ((optimize->nsimulations
01639                + optimize->nestimates * optimize->nsteps)
01640               * optimize->nvariables * sizeof (double));
01641
01642   // Calculating simulations to perform for each task
01643 #if HAVE_MPI
01644 #if DEBUG_OPTIMIZE
01645   fprintf (stderr, "optimize_open:  rank=%u ntasks=%u\n",
01646            optimize->mpi_rank, ntasks);
01647 #endif
01648   optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
01649   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
01650   if (optimize->nsteps)
01651     {
01652       optimize->nstart_climbing
01653         = optimize->mpi_rank * optimize->nestimates / ntasks;
01654       optimize->nend_climbing
01655         = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
01656     }
01657 #else
01658   optimize->nstart = 0;
01659   optimize->nend = optimize->nsimulations;
01660   if (optimize->nsteps)
01661     {
01662       optimize->nstart_climbing = 0;
01663       optimize->nend_climbing = optimize->nestimates;
01664     }
01665 #endif
01666 #if DEBUG_OPTIMIZE
01667   fprintf (stderr, "optimize_open:  nstart=%u nend=%u\n", optimize->nstart,
01668            optimize->nend);
01669 #endif
01670
01671   // Calculating simulations to perform for each thread
01672   optimize->thread
```

```
01673     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01674   for (i = 0; i <= nthreads; ++i)
01675     {
01676       optimize->thread[i] = optimize->nstart
01677         + i * (optimize->nend - optimize->nstart) / nthreads;
01678 #if DEBUG_OPTIMIZE
01679       fprintf (stderr, "optimize_open:  i=%u thread=%u\n", i,
01680                optimize->thread[i]);
01681 #endif
01682     }
01683   if (optimize->nsteps)
01684     optimize->thread_climbing = (unsigned int *)
01685       alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01686
01687   // Opening result files
01688   optimize->file_result = g_fopen (optimize->result, "w");
01689   optimize->file_variables = g_fopen (optimize->variables, "w");
01690
01691   // Performing the algorithm
01692   switch (optimize->algorithm)
01693     {
01694       // Genetic algorithm
01695     case ALGORITHM_GENETIC:
01696       optimize_genetic ();
01697       break;
01698
01699       // Iterative algorithm
01700     default:
01701       optimize_iterate ();
01702     }
01703
01704   // Getting calculation time
01705   t = g_date_time_new_now (tz);
01706   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01707   g_date_time_unref (t);
01708   g_date_time_unref (t0);
01709   g_time_zone_unref (tz);
01710   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01711   fprintf (optimize->file_result, "%s = %.6lg s\n",
01712            _("Calculation time"), optimize->calculation_time);
01713
01714   // Closing result files
01715   fclose (optimize->file_variables);
01716   fclose (optimize->file_result);
01717
01718 #if DEBUG_OPTIMIZE
01719   fprintf (stderr, "optimize_open:  end\n");
01720 #endif
01721 }
```

Here is the call graph for this function:



4.21.3.21 optimize_orthogonal()

```
static void optimize_orthogonal ( )  [static]
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 761 of file optimize.c.

```
00762 {
00763   unsigned int i, j, k, l;
00764   double e;
00765   GThread *thread[nthreads];
00766   ParallelData data[nthreads];
00767 #if DEBUG_OPTIMIZE
00768   fprintf (stderr, "optimize_orthogonal:  start\n");
00769 #endif
00770   for (i = 0; i < optimize->nsimulations; ++i)
00771     {
00772       k = i;
00773       for (j = 0; j < optimize->nvariables; ++j)
00774         {
00775           l = k % optimize->nsweeps[j];
00776           k /= optimize->nsweeps[j];
00777           e = optimize->rangemin[j];
00778           if (optimize->nsweeps[j] > 1)
00779             e += (l + gsl_rng_uniform (optimize->rng))
00780               * (optimize->rangemax[j] - optimize->rangemin[j])
00781               / optimize->nsweeps[j];
00782           optimize->value[i * optimize->nvariables + j] = e;
00783         }
00784     }
00785   optimize->nsaveds = 0;
00786   if (nthreads <= 1)
00787     optimize_sequential ();
00788   else
00789     {
00790       for (i = 0; i < nthreads; ++i)
00791         {
00792           data[i].thread = i;
00793           thread[i]
00794             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00795         }
00796       for (i = 0; i < nthreads; ++i)
00797         g_thread_join (thread[i]);
00798     }
00799 #if HAVE_MPI
00800   // Communicating tasks results
00801   optimize_synchronise ();
00802 #endif
00803 #if DEBUG_OPTIMIZE
00804   fprintf (stderr, "optimize_orthogonal:  end\n");
00805 #endif
00806 }
```

4.21.3.22 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

simulation	Simulation number.
experiment	Experiment number.

Definition at line 184 of file optimize.c.

```
00186 {
00187   unsigned int i;
00188   double e;
00189   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00190     *buffer3, *buffer4;
00191   FILE *file_result;
00192
00193 #if DEBUG_OPTIMIZE
00194   fprintf (stderr, "optimize_parse:  start\n");
00195   fprintf (stderr, "optimize_parse:  simulation=%u experiment=%u\n",
00196            simulation, experiment);
00197 #endif
00198
00199   // Opening input files
00200   for (i = 0; i < optimize->ninputs; ++i)
00201     {
00202       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00203 #if DEBUG_OPTIMIZE
00204       fprintf (stderr, "optimize_parse:  i=%u input=%s\n", i, &input[i][0]);
00205 #endif
00206       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00207     }
00208   for (; i < MAX_NINPUTS; ++i)
00209     strcpy (&input[i][0], "");
00210 #if DEBUG_OPTIMIZE
00211   fprintf (stderr, "optimize_parse:  parsing end\n");
00212 #endif
00213
00214   // Performing the simulation
00215   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00216   buffer2 = g_path_get_dirname (optimize->simulator);
00217   buffer3 = g_path_get_basename (optimize->simulator);
00218   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00219   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00220             buffer4, input[0], input[1], input[2], input[3], input[4],
00221             input[5], input[6], input[7], output);
00222   g_free (buffer4);
00223   g_free (buffer3);
00224   g_free (buffer2);
00225 #if DEBUG_OPTIMIZE
00226   fprintf (stderr, "optimize_parse:  %s\n", buffer);
00227 #endif
00228   if (system (buffer) == -1)
00229     error_message = buffer;
00230
00231   // Checking the objective value function
00232   if (optimize->evaluator)
00233     {
00234       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00235       buffer2 = g_path_get_dirname (optimize->evaluator);
00236       buffer3 = g_path_get_basename (optimize->evaluator);
00237       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00238       snprintf (buffer, 512, "\"%s\" %s %s %s",
00239                 buffer4, output, optimize->experiment[experiment], result);
00240       g_free (buffer4);
00241       g_free (buffer3);
00242       g_free (buffer2);
00243 #if DEBUG_OPTIMIZE
00244       fprintf (stderr, "optimize_parse:  %s\n", buffer);
00245       fprintf (stderr, "optimize_parse:  result=%s\n", result);
00246 #endif
00247       if (system (buffer) == -1)
00248         error_message = buffer;
00249       file_result = g_fopen (result, "r");
00250       e = atof (fgets (buffer, 512, file_result));
00251       fclose (file_result);
00252     }
00253   else
00254     {
00255 #if DEBUG_OPTIMIZE
00256       fprintf (stderr, "optimize_parse:  output=%s\n", output);
00257 #endif
00258       strcpy (result, "");
00259       file_result = g_fopen (output, "r");
00260       e = atof (fgets (buffer, 512, file_result));
00261       fclose (file_result);
00262     }
```

```
00263
00264   // Removing files
00265 #if !DEBUG_OPTIMIZE
00266   for (i = 0; i < optimize->ninputs; ++i)
00267     {
00268       if (optimize->file[i][0])
00269         {
00270           snprintf (buffer, 512, RM " %s", &input[i][0]);
00271           if (system (buffer) == -1)
00272             error_message = buffer;
00273         }
00274     }
00275   snprintf (buffer, 512, RM " %s %s", output, result);
00276   if (system (buffer) == -1)
00277     error_message = buffer;
00278 #endif
00279
00280   // Processing pending events
00281   if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE
00285   fprintf (stderr, "optimize_parse:  end\n");
00286 #endif
00287
00288   // Returning the objective function
00289   return e * optimize->weight[experiment];
00290 }
```

Here is the call graph for this function:

4.21.3.23 optimize_print()

```
static void optimize_print ( )  [static]
```

Function to print the results.

Definition at line 399 of file optimize.c.

```
00400 {
00401   unsigned int i;
00402   char buffer[512];
00403 #if HAVE_MPI
00404   if (optimize->mpi_rank)
00405     return;
00406 #endif
00407   printf ("%s\n", _("Best result"));
00408   fprintf (optimize->file_result, "%s\n", _("Best result"));
00409   printf ("error = %.15le\n", optimize->error_old[0]);
00410   fprintf (optimize->file_result, "error = %.15le\n", optimize->error_old[0]);
00411   for (i = 0; i < optimize->nvariables; ++i)
00412     {
00413       snprintf (buffer, 512, "%s = %s\n",
00414                 optimize->label[i], format[optimize->precision[i]]);
00415       printf (buffer, optimize->value_old[i]);
00416       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00417     }
00418   fflush (optimize->file_result);
00419 }
```

4.21.3.24 optimize_refine()

```
static void optimize_refine ( )  [inline], [static]
```

Function to refine the search ranges of the variables in iterative algorithms.

Definition at line 1266 of file optimize.c.

```
01267 {
01268   unsigned int i, j;
01269   double d;
01270 #if HAVE_MPI
01271   MPI_Status mpi_stat;
01272 #endif
01273 #if DEBUG_OPTIMIZE
01274   fprintf (stderr, "optimize_refine:  start\n");
01275 #endif
01276 #if HAVE_MPI
01277   if (!optimize->mpi_rank)
01278     {
01279 #endif
01280       for (j = 0; j < optimize->nvariables; ++j)
01281         {
01282           optimize->rangemin[j] = optimize->rangemax[j]
01283             = optimize->value_old[j];
01284         }
01285       for (i = 0; ++i < optimize->nbest;)
01286         {
01287           for (j = 0; j < optimize->nvariables; ++j)
01288             {
01289               optimize->rangemin[j]
01290                 = fmin (optimize->rangemin[j],
01291                         optimize->value_old[i * optimize->nvariables + j]);
01292               optimize->rangemax[j]
01293                 = fmax (optimize->rangemax[j],
01294                         optimize->value_old[i * optimize->nvariables + j]);
01295             }
01296         }
01297       for (j = 0; j < optimize->nvariables; ++j)
01298         {
01299           d = optimize->tolerance
01300             * (optimize->rangemax[j] - optimize->rangemin[j]);
01301           switch (optimize->algorithm)
01302             {
01303             case ALGORITHM_MONTE_CARLO:
01304               d *= 0.5;
01305               break;
01306             default:
01307               if (optimize->nsweeps[j] > 1)
01308                 d /= optimize->nsweeps[j] - 1;
01309               else
01310                 d = 0.;
01311             }
01312           optimize->rangemin[j] -= d;
01313           optimize->rangemin[j]
01314             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01315           optimize->rangemax[j] += d;
01316           optimize->rangemax[j]
01317             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01318           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01319                   optimize->rangemin[j], optimize->rangemax[j]);
01320           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01321                    optimize->label[j], optimize->rangemin[j],
01322                    optimize->rangemax[j]);
01323         }
01324 #if HAVE_MPI
01325       for (i = 1; (int) i < ntasks; ++i)
01326         {
01327           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01328                     1, MPI_COMM_WORLD);
01329           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01330                     1, MPI_COMM_WORLD);
01331         }
01332     }
01333   else
01334     {
01335       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01336                 MPI_COMM_WORLD, &mpi_stat);
01337       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01338                 MPI_COMM_WORLD, &mpi_stat);
01339     }
01340 #endif
01341 #if DEBUG_OPTIMIZE
01342   fprintf (stderr, "optimize_refine:  end\n");
```

```
01343 #endif
01344 }
```

4.21.3.25 optimize_save_old()

```
static void optimize_save_old ( )  [inline], [static]
```

Function to save the best results on iterative methods.

Definition at line 1186 of file optimize.c.

```
01187 {
01188   unsigned int i, j;
01189 #if DEBUG_OPTIMIZE
01190   fprintf (stderr, "optimize_save_old:  start\n");
01191   fprintf (stderr, "optimize_save_old:  nsaveds=%u\n", optimize->nsaveds);
01192 #endif
01193   memcpy (optimize->error_old, optimize->error_best,
01194           optimize->nbest * sizeof (double));
01195   for (i = 0; i < optimize->nbest; ++i)
01196     {
01197       j = optimize->simulation_best[i];
01198 #if DEBUG_OPTIMIZE
01199       fprintf (stderr, "optimize_save_old:  i=%u j=%u\n", i, j);
01200 #endif
01201       memcpy (optimize->value_old + i * optimize->nvariables,
01202               optimize->value + j * optimize->nvariables,
01203               optimize->nvariables * sizeof (double));
01204     }
01205 #if DEBUG_OPTIMIZE
01206   for (i = 0; i < optimize->nvariables; ++i)
01207     fprintf (stderr, "optimize_save_old:  best variable %u=%lg\n",
01208              i, optimize->value_old[i]);
01209   fprintf (stderr, "optimize_save_old:  end\n");
01210 #endif
01211 }
```

4.21.3.26 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

simulation	Simulation number.
error	Error value.

Definition at line 425 of file optimize.c.

```
00427 {
00428   unsigned int i;
00429   char buffer[64];
00430 #if DEBUG_OPTIMIZE
00431   fprintf (stderr, "optimize_save_variables:  start\n");
00432 #endif
00433   for (i = 0; i < optimize->nvariables; ++i)
00434     {
00435       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00436       fprintf (optimize->file_variables, buffer,
00437                optimize->value[simulation * optimize->nvariables + i]);
00438     }
```

```
00439   fprintf (optimize->file_variables, "%.14le\n", error);
00440   fflush (optimize->file_variables);
00441 #if DEBUG_OPTIMIZE
00442   fprintf (stderr, "optimize_save_variables:  end\n");
00443 #endif
00444 }
```

4.21.3.27 optimize_sequential()

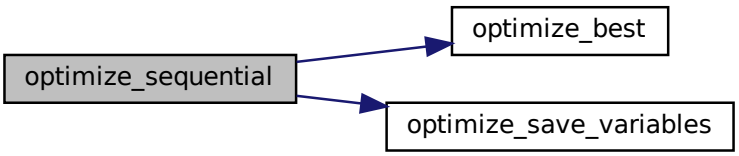
```
static void optimize_sequential ( )  [static]
```

Function to optimize sequentially.

Definition at line 491 of file optimize.c.

```
00492 {
00493   unsigned int i;
00494   double e;
00495 #if DEBUG_OPTIMIZE
00496   fprintf (stderr, "optimize_sequential:  start\n");
00497   fprintf (stderr, "optimize_sequential:  nstart=%u nend=%u\n",
00498            optimize->nstart, optimize->nend);
00499 #endif
00500   for (i = optimize->nstart; i < optimize->nend; ++i)
00501     {
00502       e = optimize_norm (i);
00503       optimize_best (i, e);
00504       optimize_save_variables (i, e);
00505       if (e < optimize->threshold)
00506         {
00507           optimize->stop = 1;
00508           break;
00509         }
00510 #if DEBUG_OPTIMIZE
00511       fprintf (stderr, "optimize_sequential:  i=%u e=%lg\n", i, e);
00512 #endif
00513     }
00514 #if DEBUG_OPTIMIZE
00515   fprintf (stderr, "optimize_sequential:  end\n");
00516 #endif
00517 }
```

Here is the call graph for this function:



4.21.3.28 optimize_step()

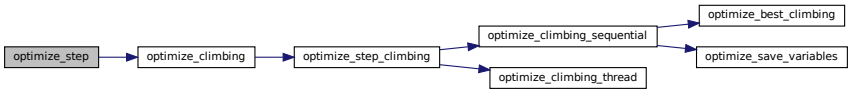
```
static void optimize_step ( )  [static]
```

Function to do a step of the iterative algorithm.

Definition at line 1350 of file optimize.c.

```
01351 {
01352 #if DEBUG_OPTIMIZE
01353   fprintf (stderr, "optimize_step:  start\n");
01354 #endif
01355   optimize_algorithm ();
01356   if (optimize->nsteps)
01357     optimize_climbing ();
01358 #if DEBUG_OPTIMIZE
01359   fprintf (stderr, "optimize_step:  end\n");
01360 #endif
01361 }
```

Here is the call graph for this function:



4.21.3.29 optimize_step_climbing()

```
static void optimize_step_climbing (
             unsigned int simulation )  [inline], [static]
```

Function to do a step of the hill climbing method.

Parameters

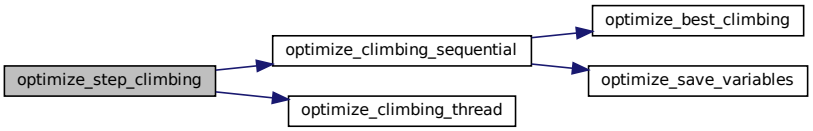
simulation	Simulation number.

Definition at line 971 of file optimize.c.

```
00972 {
00973   GThread *thread[nthreads_climbing];
00974   ParallelData data[nthreads_climbing];
00975   unsigned int i, j, k, b;
00976 #if DEBUG_OPTIMIZE
00977   fprintf (stderr, "optimize_step_climbing:  start\n");
00978 #endif
00979   for (i = 0; i < optimize->nestimates; ++i)
00980     {
00981       k = (simulation + i) * optimize->nvariables;
00982       b = optimize->simulation_best[0] * optimize->nvariables;
00983 #if DEBUG_OPTIMIZE
00984       fprintf (stderr, "optimize_step_climbing:  simulation=%u best=%u\n",
00985                simulation + i, optimize->simulation_best[0]);
00986 #endif
00987       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00988         {
00989 #if DEBUG_OPTIMIZE
00990           fprintf (stderr,
00991                    "optimize_step_climbing:  estimate=%u best%u=%.14le\n",
00992                    i, j, optimize->value[b]);
00993 #endif
```

```
00994           optimize->value[k]
00995             = optimize->value[b] + optimize_estimate_climbing (j, i);
00996           optimize->value[k] = fmin (fmax (optimize->value[k],
00997                                            optimize->rangeminabs[j]),
00998                                      optimize->rangemaxabs[j]);
00999 #if DEBUG_OPTIMIZE
01000           fprintf (stderr,
01001                    "optimize_step_climbing:  estimate=%u variable%u=%.14le\n",
01002                    i, j, optimize->value[k]);
01003 #endif
01004         }
01005     }
01006   if (nthreads_climbing == 1)
01007     optimize_climbing_sequential (simulation);
01008   else
01009     {
01010       for (i = 0; i <= nthreads_climbing; ++i)
01011         {
01012           optimize->thread_climbing[i]
01013             = simulation + optimize->nstart_climbing
01014             + i * (optimize->nend_climbing - optimize->nstart_climbing)
01015             / nthreads_climbing;
01016 #if DEBUG_OPTIMIZE
01017           fprintf (stderr,
01018                    "optimize_step_climbing:  i=%u thread_climbing=%u\n",
01019                    i, optimize->thread_climbing[i]);
01020 #endif
01021         }
01022       for (i = 0; i < nthreads_climbing; ++i)
01023         {
01024           data[i].thread = i;
01025           thread[i] = g_thread_new
01026             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01027         }
01028       for (i = 0; i < nthreads_climbing; ++i)
01029         g_thread_join (thread[i]);
01030     }
01031 #if DEBUG_OPTIMIZE
01032   fprintf (stderr, "optimize_step_climbing:  end\n");
01033 #endif
01034 }
```

Here is the call graph for this function:



4.21.3.30 optimize_sweep()

```
static void optimize_sweep ( )  [static]
```

Function to optimize with the sweep algorithm.

Definition at line 671 of file optimize.c.

```
00672 {
00673   unsigned int i, j, k, l;
00674   double e;
00675   GThread *thread[nthreads];
00676   ParallelData data[nthreads];
00677 #if DEBUG_OPTIMIZE
00678   fprintf (stderr, "optimize_sweep:  start\n");
00679 #endif
00680   for (i = 0; i < optimize->nsimulations; ++i)
```

```
00681     {
00682       k = i;
00683       for (j = 0; j < optimize->nvariables; ++j)
00684         {
00685           l = k % optimize->nsweeps[j];
00686           k /= optimize->nsweeps[j];
00687           e = optimize->rangemin[j];
00688           if (optimize->nsweeps[j] > 1)
00689             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00690               / (optimize->nsweeps[j] - 1);
00691           optimize->value[i * optimize->nvariables + j] = e;
00692         }
00693     }
00694   optimize->nsaveds = 0;
00695   if (nthreads <= 1)
00696     optimize_sequential ();
00697   else
00698     {
00699       for (i = 0; i < nthreads; ++i)
00700         {
00701           data[i].thread = i;
00702           thread[i]
00703             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00704         }
00705       for (i = 0; i < nthreads; ++i)
00706         g_thread_join (thread[i]);
00707     }
00708 #if HAVE_MPI
00709   // Communicating tasks results
00710   optimize_synchronise ();
00711 #endif
00712 #if DEBUG_OPTIMIZE
00713   fprintf (stderr, "optimize_sweep:  end\n");
00714 #endif
00715 }
```

4.21.3.31 optimize_synchronise()

```
static void optimize_synchronise ( )  [static]
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 624 of file optimize.c.

```
00625 {
00626   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00627   double error_best[optimize->nbest];
00628   MPI_Status mpi_stat;
00629 #if DEBUG_OPTIMIZE
00630   fprintf (stderr, "optimize_synchronise:  start\n");
00631 #endif
00632   if (optimize->mpi_rank == 0)
00633     {
00634       for (i = 1; (int) i < ntasks; ++i)
00635         {
00636           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00638                     MPI_COMM_WORLD, &mpi_stat);
00639           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00640                     MPI_COMM_WORLD, &mpi_stat);
00641           optimize_merge (nsaveds, simulation_best, error_best);
00642           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00643           if (stop)
00644             optimize->stop = 1;
00645         }
00646       for (i = 1; (int) i < ntasks; ++i)
00647         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00648     }
00649   else
00650     {
00651       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00652       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00653                 MPI_COMM_WORLD);
00654       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00655                 MPI_COMM_WORLD);
00656       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00657       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00658       if (stop)
00659         optimize->stop = 1;
```

```
00660     }
00661 #if DEBUG_OPTIMIZE
00662   fprintf (stderr, "optimize_synchronise:  end\n");
00663 #endif
00664 }
```

4.21.3.32 optimize_thread()

```
static void * optimize_thread (
            ParallelData * data )  [static]
```

Function to optimize on a thread.

Returns

NULL.

Parameters

data	Function data.

Definition at line 525 of file optimize.c.

```
00526 {
00527   unsigned int i, thread;
00528   double e;
00529 #if DEBUG_OPTIMIZE
00530   fprintf (stderr, "optimize_thread:  start\n");
00531 #endif
00532   thread = data->thread;
00533 #if DEBUG_OPTIMIZE
00534   fprintf (stderr, "optimize_thread:  thread=%u start=%u end=%u\n", thread,
00535            optimize->thread[thread], optimize->thread[thread + 1]);
00536 #endif
00537   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00538     {
00539       e = optimize_norm (i);
00540       g_mutex_lock (mutex);
00541       optimize_best (i, e);
00542       optimize_save_variables (i, e);
00543       if (e < optimize->threshold)
00544         optimize->stop = 1;
00545       g_mutex_unlock (mutex);
00546       if (optimize->stop)
00547         break;
00548 #if DEBUG_OPTIMIZE
00549       fprintf (stderr, "optimize_thread:  i=%u e=%lg\n", i, e);
00550 #endif
00551     }
00552 #if DEBUG_OPTIMIZE
00553   fprintf (stderr, "optimize_thread:  end\n");
00554 #endif
00555   g_thread_exit (NULL);
00556   return NULL;
00557 }
```

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 80 of file optimize.c.

4.21.4.2 optimize

```
Optimize optimize[1]
```

Optimization data.

Definition at line 79 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 83 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 85 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 88 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
00038 #define _GNU_SOURCE
00039 #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
00073 #ifdef G_OS_WIN32
00074 #define RM "del"
00075 #else
00076 #define RM "rm"
00077 #endif
00078
00079 Optimize optimize[1];
00080 unsigned int nthreads_climbing;
00082
00083 static void (*optimize_algorithm) ();
00085 static double (*optimize_estimate_climbing) (unsigned int variable,
00086                                               unsigned int estimate);
00088 static double (*optimize_norm) (unsigned int simulation);
00090
00094 static inline void
00095 optimize_input (unsigned int simulation,
00096                 char *input,
00097                 GMappedFile * stencil)
00098 {
00099   char buffer[32], value[32];
```

```
00100   GRegex *regex;
00101   FILE *file;
00102   char *buffer2, *buffer3 = NULL, *content;
00103   gsize length;
00104   unsigned int i;
00105
00106 #if DEBUG_OPTIMIZE
00107   fprintf (stderr, "optimize_input:  start\n");
00108 #endif
00109
00110   // Checking the file
00111   if (!stencil)
00112     goto optimize_input_end;
00113
00114   // Opening stencil
00115   content = g_mapped_file_get_contents (stencil);
00116   length = g_mapped_file_get_length (stencil);
00117 #if DEBUG_OPTIMIZE
00118   fprintf (stderr, "optimize_input:  length=%lu\ncontent:\n%s", length, content);
00119 #endif
00120   file = g_fopen (input, "w");
00121
00122   // Parsing stencil
00123   for (i = 0; i < optimize->nvariables; ++i)
00124     {
00125 #if DEBUG_OPTIMIZE
00126       fprintf (stderr, "optimize_input:  variable=%u\n", i);
00127 #endif
00128       snprintf (buffer, 32, "@variable%u@", i + 1);
00129       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00130                            NULL);
00131       if (i == 0)
00132         {
00133           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00134                                              optimize->label[i],
00135                                              (GRegexMatchFlags) 0, NULL);
00136 #if DEBUG_OPTIMIZE
00137           fprintf (stderr, "optimize_input:  buffer2\n%s", buffer2);
00138 #endif
00139         }
00140       else
00141         {
00142           length = strlen (buffer3);
00143           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00144                                              optimize->label[i],
00145                                              (GRegexMatchFlags) 0, NULL);
00146           g_free (buffer3);
00147         }
00148       g_regex_unref (regex);
00149       length = strlen (buffer2);
00150       snprintf (buffer, 32, "@value%u@", i + 1);
00151       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00152                            NULL);
00153       snprintf (value, 32, format[optimize->precision[i]],
00154                 optimize->value[simulation * optimize->nvariables + i]);
00155
00156 #if DEBUG_OPTIMIZE
00157       fprintf (stderr, "optimize_input:  value=%s\n", value);
00158 #endif
00159       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00160                                          (GRegexMatchFlags) 0, NULL);
00161       g_free (buffer2);
00162       g_regex_unref (regex);
00163     }
00164
00165   // Saving input file
00166   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00167   g_free (buffer3);
00168   fclose (file);
00169
00170 optimize_input_end:
00171 #if DEBUG_OPTIMIZE
00172   fprintf (stderr, "optimize_input:  end\n");
00173 #endif
00174   return;
00175 }
00176
00183 static double
00184 optimize_parse (unsigned int simulation,
00185                 unsigned int experiment)
00186 {
00187   unsigned int i;
00188   double e;
00189   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00190     *buffer3, *buffer4;
00191   FILE *file_result;
00192
```

```
00193 #if DEBUG_OPTIMIZE
00194   fprintf (stderr, "optimize_parse:  start\n");
00195   fprintf (stderr, "optimize_parse:  simulation=%u experiment=%u\n",
00196            simulation, experiment);
00197 #endif
00198
00199   // Opening input files
00200   for (i = 0; i < optimize->ninputs; ++i)
00201     {
00202       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00203 #if DEBUG_OPTIMIZE
00204       fprintf (stderr, "optimize_parse:  i=%u input=%s\n", i, &input[i][0]);
00205 #endif
00206       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00207     }
00208   for (; i < MAX_NINPUTS; ++i)
00209     strcpy (&input[i][0], "");
00210 #if DEBUG_OPTIMIZE
00211   fprintf (stderr, "optimize_parse:  parsing end\n");
00212 #endif
00213
00214   // Performing the simulation
00215   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00216   buffer2 = g_path_get_dirname (optimize->simulator);
00217   buffer3 = g_path_get_basename (optimize->simulator);
00218   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00219   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00220             buffer4, input[0], input[1], input[2], input[3], input[4],
00221             input[5], input[6], input[7], output);
00222   g_free (buffer4);
00223   g_free (buffer3);
00224   g_free (buffer2);
00225 #if DEBUG_OPTIMIZE
00226   fprintf (stderr, "optimize_parse:  %s\n", buffer);
00227 #endif
00228   if (system (buffer) == -1)
00229     error_message = buffer;
00230
00231   // Checking the objective value function
00232   if (optimize->evaluator)
00233     {
00234       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00235       buffer2 = g_path_get_dirname (optimize->evaluator);
00236       buffer3 = g_path_get_basename (optimize->evaluator);
00237       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00238       snprintf (buffer, 512, "\"%s\" %s %s %s",
00239                 buffer4, output, optimize->experiment[experiment], result);
00240       g_free (buffer4);
00241       g_free (buffer3);
00242       g_free (buffer2);
00243 #if DEBUG_OPTIMIZE
00244       fprintf (stderr, "optimize_parse:  %s\n", buffer);
00245       fprintf (stderr, "optimize_parse:  result=%s\n", result);
00246 #endif
00247       if (system (buffer) == -1)
00248         error_message = buffer;
00249       file_result = g_fopen (result, "r");
00250       e = atof (fgets (buffer, 512, file_result));
00251       fclose (file_result);
00252     }
00253   else
00254     {
00255 #if DEBUG_OPTIMIZE
00256       fprintf (stderr, "optimize_parse:  output=%s\n", output);
00257 #endif
00258       strcpy (result, "");
00259       file_result = g_fopen (output, "r");
00260       e = atof (fgets (buffer, 512, file_result));
00261       fclose (file_result);
00262     }
00263
00264   // Removing files
00265 #if !DEBUG_OPTIMIZE
00266   for (i = 0; i < optimize->ninputs; ++i)
00267     {
00268       if (optimize->file[i][0])
00269         {
00270           snprintf (buffer, 512, RM " %s", &input[i][0]);
00271           if (system (buffer) == -1)
00272             error_message = buffer;
00273         }
00274     }
00275   snprintf (buffer, 512, RM " %s %s", output, result);
00276   if (system (buffer) == -1)
00277     error_message = buffer;
00278 #endif
00279
```

```
00280   // Processing pending events
00281   if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE
00285   fprintf (stderr, "optimize_parse:  end\n");
00286 #endif
00287
00288   // Returning the objective function
00289   return e * optimize->weight[experiment];
00290 }
00291
00297 static double
00298 optimize_norm_euclidian (unsigned int simulation)
00299 {
00300   double e, ei;
00301   unsigned int i;
00302 #if DEBUG_OPTIMIZE
00303   fprintf (stderr, "optimize_norm_euclidian:  start\n");
00304 #endif
00305   e = 0.;
00306   for (i = 0; i < optimize->nexperiments; ++i)
00307     {
00308       ei = optimize_parse (simulation, i);
00309       e += ei * ei;
00310     }
00311   e = sqrt (e);
00312 #if DEBUG_OPTIMIZE
00313   fprintf (stderr, "optimize_norm_euclidian:  error=%lg\n", e);
00314   fprintf (stderr, "optimize_norm_euclidian:  end\n");
00315 #endif
00316   return e;
00317 }
00318
00324 static double
00325 optimize_norm_maximum (unsigned int simulation)
00326 {
00327   double e, ei;
00328   unsigned int i;
00329 #if DEBUG_OPTIMIZE
00330   fprintf (stderr, "optimize_norm_maximum:  start\n");
00331 #endif
00332   e = 0.;
00333   for (i = 0; i < optimize->nexperiments; ++i)
00334     {
00335       ei = fabs (optimize_parse (simulation, i));
00336       e = fmax (e, ei);
00337     }
00338 #if DEBUG_OPTIMIZE
00339   fprintf (stderr, "optimize_norm_maximum:  error=%lg\n", e);
00340   fprintf (stderr, "optimize_norm_maximum:  end\n");
00341 #endif
00342   return e;
00343 }
00344
00350 static double
00351 optimize_norm_p (unsigned int simulation)
00352 {
00353   double e, ei;
00354   unsigned int i;
00355 #if DEBUG_OPTIMIZE
00356   fprintf (stderr, "optimize_norm_p:  start\n");
00357 #endif
00358   e = 0.;
00359   for (i = 0; i < optimize->nexperiments; ++i)
00360     {
00361       ei = fabs (optimize_parse (simulation, i));
00362       e += pow (ei, optimize->p);
00363     }
00364   e = pow (e, 1.  / optimize->p);
00365 #if DEBUG_OPTIMIZE
00366   fprintf (stderr, "optimize_norm_p:  error=%lg\n", e);
00367   fprintf (stderr, "optimize_norm_p:  end\n");
00368 #endif
00369   return e;
00370 }
00371
00377 static double
00378 optimize_norm_taxicab (unsigned int simulation)
00379 {
00380   double e;
00381   unsigned int i;
00382 #if DEBUG_OPTIMIZE
00383   fprintf (stderr, "optimize_norm_taxicab:  start\n");
00384 #endif
00385   e = 0.;
00386   for (i = 0; i < optimize->nexperiments; ++i)
```

```
00387     e += fabs (optimize_parse (simulation, i));
00388 #if DEBUG_OPTIMIZE
00389   fprintf (stderr, "optimize_norm_taxicab:  error=%lg\n", e);
00390   fprintf (stderr, "optimize_norm_taxicab:  end\n");
00391 #endif
00392   return e;
00393 }
00394
00398 static void
00399 optimize_print ()
00400 {
00401   unsigned int i;
00402   char buffer[512];
00403 #if HAVE_MPI
00404   if (optimize->mpi_rank)
00405     return;
00406 #endif
00407   printf ("%s\n", _("Best result"));
00408   fprintf (optimize->file_result, "%s\n", _("Best result"));
00409   printf ("error = %.15le\n", optimize->error_old[0]);
00410   fprintf (optimize->file_result, "error = %.15le\n", optimize->error_old[0]);
00411   for (i = 0; i < optimize->nvariables; ++i)
00412     {
00413       snprintf (buffer, 512, "%s = %s\n",
00414                 optimize->label[i], format[optimize->precision[i]]);
00415       printf (buffer, optimize->value_old[i]);
00416       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00417     }
00418   fflush (optimize->file_result);
00419 }
00420
00424 static void
00425 optimize_save_variables (unsigned int simulation,
00426                          double error)
00427 {
00428   unsigned int i;
00429   char buffer[64];
00430 #if DEBUG_OPTIMIZE
00431   fprintf (stderr, "optimize_save_variables:  start\n");
00432 #endif
00433   for (i = 0; i < optimize->nvariables; ++i)
00434     {
00435       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00436       fprintf (optimize->file_variables, buffer,
00437                optimize->value[simulation * optimize->nvariables + i]);
00438     }
00439   fprintf (optimize->file_variables, "%.14le\n", error);
00440   fflush (optimize->file_variables);
00441 #if DEBUG_OPTIMIZE
00442   fprintf (stderr, "optimize_save_variables:  end\n");
00443 #endif
00444 }
00445
00449 static void
00450 optimize_best (unsigned int simulation,
00451                double value)
00452 {
00453   unsigned int i, j;
00454   double e;
00455 #if DEBUG_OPTIMIZE
00456   fprintf (stderr, "optimize_best:  start\n");
00457   fprintf (stderr, "optimize_best:  nsaveds=%u nbest=%u\n",
00458            optimize->nsaveds, optimize->nbest);
00459 #endif
00460   if (optimize->nsaveds < optimize->nbest
00461       || value < optimize->error_best[optimize->nsaveds - 1])
00462     {
00463       if (optimize->nsaveds < optimize->nbest)
00464         ++optimize->nsaveds;
00465       optimize->error_best[optimize->nsaveds - 1] = value;
00466       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00467       for (i = optimize->nsaveds; --i;)
00468         {
00469           if (optimize->error_best[i] < optimize->error_best[i - 1])
00470             {
00471               j = optimize->simulation_best[i];
00472               e = optimize->error_best[i];
00473               optimize->simulation_best[i] = optimize->simulation_best[i - 1];
00474               optimize->error_best[i] = optimize->error_best[i - 1];
00475               optimize->simulation_best[i - 1] = j;
00476               optimize->error_best[i - 1] = e;
00477             }
00478           else
00479             break;
00480         }
00481     }
00482 #if DEBUG_OPTIMIZE
```

```
00483   fprintf (stderr, "optimize_best:  end\n");
00484 #endif
00485 }
00486
00490 static void
00491 optimize_sequential ()
00492 {
00493   unsigned int i;
00494   double e;
00495 #if DEBUG_OPTIMIZE
00496   fprintf (stderr, "optimize_sequential:  start\n");
00497   fprintf (stderr, "optimize_sequential:  nstart=%u nend=%u\n",
00498            optimize->nstart, optimize->nend);
00499 #endif
00500   for (i = optimize->nstart; i < optimize->nend; ++i)
00501     {
00502       e = optimize_norm (i);
00503       optimize_best (i, e);
00504       optimize_save_variables (i, e);
00505       if (e < optimize->threshold)
00506         {
00507           optimize->stop = 1;
00508           break;
00509         }
00510 #if DEBUG_OPTIMIZE
00511       fprintf (stderr, "optimize_sequential:  i=%u e=%lg\n", i, e);
00512 #endif
00513     }
00514 #if DEBUG_OPTIMIZE
00515   fprintf (stderr, "optimize_sequential:  end\n");
00516 #endif
00517 }
00518
00524 static void *
00525 optimize_thread (ParallelData * data)
00526 {
00527   unsigned int i, thread;
00528   double e;
00529 #if DEBUG_OPTIMIZE
00530   fprintf (stderr, "optimize_thread:  start\n");
00531 #endif
00532   thread = data->thread;
00533 #if DEBUG_OPTIMIZE
00534   fprintf (stderr, "optimize_thread:  thread=%u start=%u end=%u\n", thread,
00535            optimize->thread[thread], optimize->thread[thread + 1]);
00536 #endif
00537   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00538     {
00539       e = optimize_norm (i);
00540       g_mutex_lock (mutex);
00541       optimize_best (i, e);
00542       optimize_save_variables (i, e);
00543       if (e < optimize->threshold)
00544         optimize->stop = 1;
00545       g_mutex_unlock (mutex);
00546       if (optimize->stop)
00547         break;
00548 #if DEBUG_OPTIMIZE
00549       fprintf (stderr, "optimize_thread:  i=%u e=%lg\n", i, e);
00550 #endif
00551     }
00552 #if DEBUG_OPTIMIZE
00553   fprintf (stderr, "optimize_thread:  end\n");
00554 #endif
00555   g_thread_exit (NULL);
00556   return NULL;
00557 }
00558
00562 static inline void
00563 optimize_merge (unsigned int nsaveds,
00564                 unsigned int *simulation_best,
00566                 double *error_best)
00568 {
00569   unsigned int i, j, k, s[optimize->nbest];
00570   double e[optimize->nbest];
00571 #if DEBUG_OPTIMIZE
00572   fprintf (stderr, "optimize_merge:  start\n");
00573 #endif
00574   i = j = k = 0;
00575   do
00576     {
00577       if (i == optimize->nsaveds)
00578         {
00579           s[k] = simulation_best[j];
00580           e[k] = error_best[j];
00581           ++j;
00582           ++k;
```

```
00583           if (j == nsaveds)
00584             break;
00585         }
00586       else if (j == nsaveds)
00587         {
00588           s[k] = optimize->simulation_best[i];
00589           e[k] = optimize->error_best[i];
00590           ++i;
00591           ++k;
00592           if (i == optimize->nsaveds)
00593             break;
00594         }
00595       else if (optimize->error_best[i] > error_best[j])
00596         {
00597           s[k] = simulation_best[j];
00598           e[k] = error_best[j];
00599           ++j;
00600           ++k;
00601         }
00602       else
00603         {
00604           s[k] = optimize->simulation_best[i];
00605           e[k] = optimize->error_best[i];
00606           ++i;
00607           ++k;
00608         }
00609     }
00610   while (k < optimize->nbest);
00611   optimize->nsaveds = k;
00612   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00613   memcpy (optimize->error_best, e, k * sizeof (double));
00614 #if DEBUG_OPTIMIZE
00615   fprintf (stderr, "optimize_merge:  end\n");
00616 #endif
00617 }
00618
00622 #if HAVE_MPI
00623 static void
00624 optimize_synchronise ()
00625 {
00626   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00627   double error_best[optimize->nbest];
00628   MPI_Status mpi_stat;
00629 #if DEBUG_OPTIMIZE
00630   fprintf (stderr, "optimize_synchronise:  start\n");
00631 #endif
00632   if (optimize->mpi_rank == 0)
00633     {
00634       for (i = 1; (int) i < ntasks; ++i)
00635         {
00636           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00638                     MPI_COMM_WORLD, &mpi_stat);
00639           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00640                     MPI_COMM_WORLD, &mpi_stat);
00641           optimize_merge (nsaveds, simulation_best, error_best);
00642           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00643           if (stop)
00644             optimize->stop = 1;
00645         }
00646       for (i = 1; (int) i < ntasks; ++i)
00647         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00648     }
00649   else
00650     {
00651       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00652       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00653                 MPI_COMM_WORLD);
00654       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00655                 MPI_COMM_WORLD);
00656       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00657       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00658       if (stop)
00659         optimize->stop = 1;
00660     }
00661 #if DEBUG_OPTIMIZE
00662   fprintf (stderr, "optimize_synchronise:  end\n");
00663 #endif
00664 }
00665 #endif
00666
00670 static void
00671 optimize_sweep ()
00672 {
00673   unsigned int i, j, k, l;
00674   double e;
00675   GThread *thread[nthreads];
```

```
00676   ParallelData data[nthreads];
00677 #if DEBUG_OPTIMIZE
00678   fprintf (stderr, "optimize_sweep:  start\n");
00679 #endif
00680   for (i = 0; i < optimize->nsimulations; ++i)
00681     {
00682       k = i;
00683       for (j = 0; j < optimize->nvariables; ++j)
00684         {
00685           l = k % optimize->nsweeps[j];
00686           k /= optimize->nsweeps[j];
00687           e = optimize->rangemin[j];
00688           if (optimize->nsweeps[j] > 1)
00689             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00690               / (optimize->nsweeps[j] - 1);
00691           optimize->value[i * optimize->nvariables + j] = e;
00692         }
00693     }
00694   optimize->nsaveds = 0;
00695   if (nthreads <= 1)
00696     optimize_sequential ();
00697   else
00698     {
00699       for (i = 0; i < nthreads; ++i)
00700         {
00701           data[i].thread = i;
00702           thread[i]
00703             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00704         }
00705       for (i = 0; i < nthreads; ++i)
00706         g_thread_join (thread[i]);
00707     }
00708 #if HAVE_MPI
00709   // Communicating tasks results
00710   optimize_synchronise ();
00711 #endif
00712 #if DEBUG_OPTIMIZE
00713   fprintf (stderr, "optimize_sweep:  end\n");
00714 #endif
00715 }
00716
00720 static void
00721 optimize_MonteCarlo ()
00722 {
00723   unsigned int i, j;
00724   GThread *thread[nthreads];
00725   ParallelData data[nthreads];
00726 #if DEBUG_OPTIMIZE
00727   fprintf (stderr, "optimize_MonteCarlo:  start\n");
00728 #endif
00729   for (i = 0; i < optimize->nsimulations; ++i)
00730     for (j = 0; j < optimize->nvariables; ++j)
00731       optimize->value[i * optimize->nvariables + j]
00732         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00733         * (optimize->rangemax[j] - optimize->rangemin[j]);
00734   optimize->nsaveds = 0;
00735   if (nthreads <= 1)
00736     optimize_sequential ();
00737   else
00738     {
00739       for (i = 0; i < nthreads; ++i)
00740         {
00741           data[i].thread = i;
00742           thread[i]
00743             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00744         }
00745       for (i = 0; i < nthreads; ++i)
00746         g_thread_join (thread[i]);
00747     }
00748 #if HAVE_MPI
00749   // Communicating tasks results
00750   optimize_synchronise ();
00751 #endif
00752 #if DEBUG_OPTIMIZE
00753   fprintf (stderr, "optimize_MonteCarlo:  end\n");
00754 #endif
00755 }
00756
00760 static void
00761 optimize_orthogonal ()
00762 {
00763   unsigned int i, j, k, l;
00764   double e;
00765   GThread *thread[nthreads];
00766   ParallelData data[nthreads];
00767 #if DEBUG_OPTIMIZE
00768   fprintf (stderr, "optimize_orthogonal:  start\n");
```

```
00769 #endif
00770   for (i = 0; i < optimize->nsimulations; ++i)
00771     {
00772       k = i;
00773       for (j = 0; j < optimize->nvariables; ++j)
00774         {
00775           l = k % optimize->nsweeps[j];
00776           k /= optimize->nsweeps[j];
00777           e = optimize->rangemin[j];
00778           if (optimize->nsweeps[j] > 1)
00779             e += (l + gsl_rng_uniform (optimize->rng))
00780               * (optimize->rangemax[j] - optimize->rangemin[j])
00781               / optimize->nsweeps[j];
00782           optimize->value[i * optimize->nvariables + j] = e;
00783         }
00784     }
00785   optimize->nsaveds = 0;
00786   if (nthreads <= 1)
00787     optimize_sequential ();
00788   else
00789     {
00790       for (i = 0; i < nthreads; ++i)
00791         {
00792           data[i].thread = i;
00793           thread[i]
00794             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00795         }
00796       for (i = 0; i < nthreads; ++i)
00797         g_thread_join (thread[i]);
00798     }
00799 #if HAVE_MPI
00800   // Communicating tasks results
00801   optimize_synchronise ();
00802 #endif
00803 #if DEBUG_OPTIMIZE
00804   fprintf (stderr, "optimize_orthogonal:  end\n");
00805 #endif
00806 }
00807
00811 static void
00812 optimize_best_climbing (unsigned int simulation,
00813                         double value)
00814 {
00815 #if DEBUG_OPTIMIZE
00816   fprintf (stderr, "optimize_best_climbing:  start\n");
00817   fprintf (stderr,
00818            "optimize_best_climbing:  simulation=%u value=%.14le best=%.14le\n",
00819            simulation, value, optimize->error_best[0]);
00820 #endif
00821   if (value < optimize->error_best[0])
00822     {
00823       optimize->error_best[0] = value;
00824       optimize->simulation_best[0] = simulation;
00825 #if DEBUG_OPTIMIZE
00826       fprintf (stderr,
00827                "optimize_best_climbing:  BEST simulation=%u value=%.14le\n",
00828                simulation, value);
00829 #endif
00830     }
00831 #if DEBUG_OPTIMIZE
00832   fprintf (stderr, "optimize_best_climbing:  end\n");
00833 #endif
00834 }
00835
00839 static inline void
00840 optimize_climbing_sequential (unsigned int simulation)
00841 {
00842   double e;
00843   unsigned int i, j;
00844 #if DEBUG_OPTIMIZE
00845   fprintf (stderr, "optimize_climbing_sequential:  start\n");
00846   fprintf (stderr, "optimize_climbing_sequential:  nstart_climbing=%u "
00847            "nend_climbing=%u\n",
00848            optimize->nstart_climbing, optimize->nend_climbing);
00849 #endif
00850   for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00851     {
00852       j = simulation + i;
00853       e = optimize_norm (j);
00854       optimize_best_climbing (j, e);
00855       optimize_save_variables (j, e);
00856       if (e < optimize->threshold)
00857         {
00858           optimize->stop = 1;
00859           break;
00860         }
00861 #if DEBUG_OPTIMIZE
```

```
00862       fprintf (stderr, "optimize_climbing_sequential:  i=%u e=%lg\n", i, e);
00863 #endif
00864     }
00865 #if DEBUG_OPTIMIZE
00866   fprintf (stderr, "optimize_climbing_sequential:  end\n");
00867 #endif
00868 }
00869
00875 static void *
00876 optimize_climbing_thread (ParallelData * data)
00877 {
00878   unsigned int i, thread;
00879   double e;
00880 #if DEBUG_OPTIMIZE
00881   fprintf (stderr, "optimize_climbing_thread:  start\n");
00882 #endif
00883   thread = data->thread;
00884 #if DEBUG_OPTIMIZE
00885   fprintf (stderr, "optimize_climbing_thread:  thread=%u start=%u end=%u\n",
00886            thread,
00887            optimize->thread_climbing[thread],
00888            optimize->thread_climbing[thread + 1]);
00889 #endif
00890   for (i = optimize->thread_climbing[thread];
00891        i < optimize->thread_climbing[thread + 1]; ++i)
00892     {
00893       e = optimize_norm (i);
00894       g_mutex_lock (mutex);
00895       optimize_best_climbing (i, e);
00896       optimize_save_variables (i, e);
00897       if (e < optimize->threshold)
00898         optimize->stop = 1;
00899       g_mutex_unlock (mutex);
00900       if (optimize->stop)
00901         break;
00902 #if DEBUG_OPTIMIZE
00903       fprintf (stderr, "optimize_climbing_thread:  i=%u e=%lg\n", i, e);
00904 #endif
00905     }
00906 #if DEBUG_OPTIMIZE
00907   fprintf (stderr, "optimize_climbing_thread:  end\n");
00908 #endif
00909   g_thread_exit (NULL);
00910   return NULL;
00911 }
00912
00916 static double
00917 optimize_estimate_climbing_random (unsigned int variable,
00919                                    unsigned int estimate
00920                                    __attribute__((unused)))
00922 {
00923   double x;
00924 #if DEBUG_OPTIMIZE
00925   fprintf (stderr, "optimize_estimate_climbing_random:  start\n");
00926 #endif
00927   x = optimize->climbing[variable]
00928     + (1.  - 2.  * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00929 #if DEBUG_OPTIMIZE
00930   fprintf (stderr, "optimize_estimate_climbing_random:  climbing%u=%lg\n",
00931            variable, x);
00932   fprintf (stderr, "optimize_estimate_climbing_random:  end\n");
00933 #endif
00934   return x;
00935 }
00936
00940 static double
00941 optimize_estimate_climbing_coordinates (unsigned int variable,
00943                                         unsigned int estimate)
00945 {
00946   double x;
00947 #if DEBUG_OPTIMIZE
00948   fprintf (stderr, "optimize_estimate_climbing_coordinates:  start\n");
00949 #endif
00950   x = optimize->climbing[variable];
00951   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00952     {
00953       if (estimate & 1)
00954         x += optimize->step[variable];
00955       else
00956         x -= optimize->step[variable];
00957     }
00958 #if DEBUG_OPTIMIZE
00959   fprintf (stderr,
00960            "optimize_estimate_climbing_coordinates:  climbing%u=%lg\n",
00961            variable, x);
00962   fprintf (stderr, "optimize_estimate_climbing_coordinates:  end\n");
00963 #endif
```

```
00964   return x;
00965 }
00966
00970 static inline void
00971 optimize_step_climbing (unsigned int simulation)
00972 {
00973   GThread *thread[nthreads_climbing];
00974   ParallelData data[nthreads_climbing];
00975   unsigned int i, j, k, b;
00976 #if DEBUG_OPTIMIZE
00977   fprintf (stderr, "optimize_step_climbing:  start\n");
00978 #endif
00979   for (i = 0; i < optimize->nestimates; ++i)
00980     {
00981       k = (simulation + i) * optimize->nvariables;
00982       b = optimize->simulation_best[0] * optimize->nvariables;
00983 #if DEBUG_OPTIMIZE
00984       fprintf (stderr, "optimize_step_climbing:  simulation=%u best=%u\n",
00985                simulation + i, optimize->simulation_best[0]);
00986 #endif
00987       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00988         {
00989 #if DEBUG_OPTIMIZE
00990           fprintf (stderr,
00991                    "optimize_step_climbing:  estimate=%u best%u=%.14le\n",
00992                    i, j, optimize->value[b]);
00993 #endif
00994           optimize->value[k]
00995             = optimize->value[b] + optimize_estimate_climbing (j, i);
00996           optimize->value[k] = fmin (fmax (optimize->value[k],
00997                                            optimize->rangeminabs[j]),
00998                                      optimize->rangemaxabs[j]);
00999 #if DEBUG_OPTIMIZE
01000           fprintf (stderr,
01001                    "optimize_step_climbing:  estimate=%u variable%u=%.14le\n",
01002                    i, j, optimize->value[k]);
01003 #endif
01004         }
01005     }
01006   if (nthreads_climbing == 1)
01007     optimize_climbing_sequential (simulation);
01008   else
01009     {
01010       for (i = 0; i <= nthreads_climbing; ++i)
01011         {
01012           optimize->thread_climbing[i]
01013             = simulation + optimize->nstart_climbing
01014             + i * (optimize->nend_climbing - optimize->nstart_climbing)
01015             / nthreads_climbing;
01016 #if DEBUG_OPTIMIZE
01017           fprintf (stderr,
01018                    "optimize_step_climbing:  i=%u thread_climbing=%u\n",
01019                    i, optimize->thread_climbing[i]);
01020 #endif
01021         }
01022       for (i = 0; i < nthreads_climbing; ++i)
01023         {
01024           data[i].thread = i;
01025           thread[i] = g_thread_new
01026             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01027         }
01028       for (i = 0; i < nthreads_climbing; ++i)
01029         g_thread_join (thread[i]);
01030     }
01031 #if DEBUG_OPTIMIZE
01032   fprintf (stderr, "optimize_step_climbing:  end\n");
01033 #endif
01034 }
01035
01039 static inline void
01040 optimize_climbing ()
01041 {
01042   unsigned int i, j, k, b, s, adjust;
01043 #if DEBUG_OPTIMIZE
01044   fprintf (stderr, "optimize_climbing:  start\n");
01045 #endif
01046   for (i = 0; i < optimize->nvariables; ++i)
01047     optimize->climbing[i] = 0.;
01048   b = optimize->simulation_best[0] * optimize->nvariables;
01049   s = optimize->nsimulations;
01050   adjust = 1;
01051   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01052     {
01053 #if DEBUG_OPTIMIZE
01054       fprintf (stderr, "optimize_climbing:  step=%u old_best=%u\n",
01055                i, optimize->simulation_best[0]);
01056 #endif
```

```
01057       optimize_step_climbing (s);
01058       k = optimize->simulation_best[0] * optimize->nvariables;
01059 #if DEBUG_OPTIMIZE
01060       fprintf (stderr, "optimize_climbing:  step=%u best=%u\n",
01061                i, optimize->simulation_best[0]);
01062 #endif
01063       if (k == b)
01064         {
01065           if (adjust)
01066             for (j = 0; j < optimize->nvariables; ++j)
01067               optimize->step[j] *= 0.5;
01068           for (j = 0; j < optimize->nvariables; ++j)
01069             optimize->climbing[j] = 0.;
01070           adjust = 1;
01071         }
01072       else
01073         {
01074           for (j = 0; j < optimize->nvariables; ++j)
01075             {
01076 #if DEBUG_OPTIMIZE
01077               fprintf (stderr,
01078                        "optimize_climbing:  best%u=%.14le old%u=%.14le\n",
01079                        j, optimize->value[k + j], j, optimize->value[b + j]);
01080 #endif
01081               optimize->climbing[j]
01082                 = (1.  - optimize->relaxation) * optimize->climbing[j]
01083                 + optimize->relaxation
01084                 * (optimize->value[k + j] - optimize->value[b + j]);
01085 #if DEBUG_OPTIMIZE
01086               fprintf (stderr, "optimize_climbing:  climbing%u=%.14le\n",
01087                        j, optimize->climbing[j]);
01088 #endif
01089             }
01090           adjust = 0;
01091         }
01092     }
01093 #if DEBUG_OPTIMIZE
01094   fprintf (stderr, "optimize_climbing:  end\n");
01095 #endif
01096 }
01097
01103 static double
01104 optimize_genetic_objective (Entity * entity)
01105 {
01106   unsigned int j;
01107   double objective;
01108   char buffer[64];
01109 #if DEBUG_OPTIMIZE
01110   fprintf (stderr, "optimize_genetic_objective:  start\n");
01111 #endif
01112   for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114       optimize->value[entity->id * optimize->nvariables + j]
01115         = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117   objective = optimize_norm (entity->id);
01118   g_mutex_lock (mutex);
01119   for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122       fprintf (optimize->file_variables, buffer,
01123                genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125   fprintf (optimize->file_variables, "%.14le\n", objective);
01126   g_mutex_unlock (mutex);
01127 #if DEBUG_OPTIMIZE
01128   fprintf (stderr, "optimize_genetic_objective:  end\n");
01129 #endif
01130   return objective;
01131 }
01132
01136 static void
01137 optimize_genetic ()
01138 {
01139   double *best_variable = NULL;
01140   char *best_genome = NULL;
01141   double best_objective = 0.;
01142 #if DEBUG_OPTIMIZE
01143   fprintf (stderr, "optimize_genetic:  start\n");
01144   fprintf (stderr, "optimize_genetic:  ntasks=%u nthreads=%u\n", ntasks,
01145            nthreads);
01146   fprintf (stderr,
01147            "optimize_genetic:  nvariables=%u population=%u generations=%u\n",
01148            optimize->nvariables, optimize->nsimulations, optimize->niterations);
01149   fprintf (stderr,
01150            "optimize_genetic:  mutation=%lg reproduction=%lg adaptation=%lg\n",
01151            optimize->mutation_ratio, optimize->reproduction_ratio,
```

```
01152           optimize->adaptation_ratio);
01153 #endif
01154   genetic_algorithm_default (optimize->nvariables,
01155                              optimize->genetic_variable,
01156                              optimize->nsimulations,
01157                              optimize->niterations,
01158                              optimize->mutation_ratio,
01159                              optimize->reproduction_ratio,
01160                              optimize->adaptation_ratio,
01161                              optimize->seed,
01162                              optimize->threshold,
01163                              &optimize_genetic_objective,
01164                              &best_genome, &best_variable, &best_objective);
01165 #if DEBUG_OPTIMIZE
01166   fprintf (stderr, "optimize_genetic:  the best\n");
01167 #endif
01168   optimize->error_old = (double *) g_malloc (sizeof (double));
01169   optimize->value_old
01170     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01171   optimize->error_old[0] = best_objective;
01172   memcpy (optimize->value_old, best_variable,
01173           optimize->nvariables * sizeof (double));
01174   g_free (best_genome);
01175   g_free (best_variable);
01176   optimize_print ();
01177 #if DEBUG_OPTIMIZE
01178   fprintf (stderr, "optimize_genetic:  end\n");
01179 #endif
01180 }
01181
01185 static inline void
01186 optimize_save_old ()
01187 {
01188   unsigned int i, j;
01189 #if DEBUG_OPTIMIZE
01190   fprintf (stderr, "optimize_save_old:  start\n");
01191   fprintf (stderr, "optimize_save_old:  nsaveds=%u\n", optimize->nsaveds);
01192 #endif
01193   memcpy (optimize->error_old, optimize->error_best,
01194           optimize->nbest * sizeof (double));
01195   for (i = 0; i < optimize->nbest; ++i)
01196     {
01197       j = optimize->simulation_best[i];
01198 #if DEBUG_OPTIMIZE
01199       fprintf (stderr, "optimize_save_old:  i=%u j=%u\n", i, j);
01200 #endif
01201       memcpy (optimize->value_old + i * optimize->nvariables,
01202               optimize->value + j * optimize->nvariables,
01203               optimize->nvariables * sizeof (double));
01204     }
01205 #if DEBUG_OPTIMIZE
01206   for (i = 0; i < optimize->nvariables; ++i)
01207     fprintf (stderr, "optimize_save_old:  best variable %u=%lg\n",
01208              i, optimize->value_old[i]);
01209   fprintf (stderr, "optimize_save_old:  end\n");
01210 #endif
01211 }
01212
01217 static inline void
01218 optimize_merge_old ()
01219 {
01220   unsigned int i, j, k;
01221   double v[optimize->nbest * optimize->nvariables], e[optimize->nbest],
01222     *enew, *eold;
01223 #if DEBUG_OPTIMIZE
01224   fprintf (stderr, "optimize_merge_old:  start\n");
01225 #endif
01226   enew = optimize->error_best;
01227   eold = optimize->error_old;
01228   i = j = k = 0;
01229   do
01230     {
01231       if (*enew < *eold)
01232         {
01233           memcpy (v + k * optimize->nvariables,
01234                   optimize->value
01235                   + optimize->simulation_best[i] * optimize->nvariables,
01236                   optimize->nvariables * sizeof (double));
01237           e[k] = *enew;
01238           ++k;
01239           ++enew;
01240           ++i;
01241         }
01242       else
01243         {
01244           memcpy (v + k * optimize->nvariables,
01245                   optimize->value_old + j * optimize->nvariables,
```

```
01246                   optimize->nvariables * sizeof (double));
01247           e[k] = *eold;
01248           ++k;
01249           ++eold;
01250           ++j;
01251         }
01252     }
01253   while (k < optimize->nbest);
01254   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01255   memcpy (optimize->error_old, e, k * sizeof (double));
01256 #if DEBUG_OPTIMIZE
01257   fprintf (stderr, "optimize_merge_old:  end\n");
01258 #endif
01259 }
01260
01265 static inline void
01266 optimize_refine ()
01267 {
01268   unsigned int i, j;
01269   double d;
01270 #if HAVE_MPI
01271   MPI_Status mpi_stat;
01272 #endif
01273 #if DEBUG_OPTIMIZE
01274   fprintf (stderr, "optimize_refine:  start\n");
01275 #endif
01276 #if HAVE_MPI
01277   if (!optimize->mpi_rank)
01278     {
01279 #endif
01280       for (j = 0; j < optimize->nvariables; ++j)
01281         {
01282           optimize->rangemin[j] = optimize->rangemax[j]
01283             = optimize->value_old[j];
01284         }
01285       for (i = 0; ++i < optimize->nbest;)
01286         {
01287           for (j = 0; j < optimize->nvariables; ++j)
01288             {
01289               optimize->rangemin[j]
01290                 = fmin (optimize->rangemin[j],
01291                         optimize->value_old[i * optimize->nvariables + j]);
01292               optimize->rangemax[j]
01293                 = fmax (optimize->rangemax[j],
01294                         optimize->value_old[i * optimize->nvariables + j]);
01295             }
01296         }
01297       for (j = 0; j < optimize->nvariables; ++j)
01298         {
01299           d = optimize->tolerance
01300             * (optimize->rangemax[j] - optimize->rangemin[j]);
01301           switch (optimize->algorithm)
01302             {
01303             case ALGORITHM_MONTE_CARLO:
01304               d *= 0.5;
01305               break;
01306             default:
01307               if (optimize->nsweeps[j] > 1)
01308                 d /= optimize->nsweeps[j] - 1;
01309               else
01310                 d = 0.;
01311             }
01312           optimize->rangemin[j] -= d;
01313           optimize->rangemin[j]
01314             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01315           optimize->rangemax[j] += d;
01316           optimize->rangemax[j]
01317             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01318           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01319                   optimize->rangemin[j], optimize->rangemax[j]);
01320           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01321                    optimize->label[j], optimize->rangemin[j],
01322                    optimize->rangemax[j]);
01323         }
01324 #if HAVE_MPI
01325       for (i = 1; (int) i < ntasks; ++i)
01326         {
01327           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01328                     1, MPI_COMM_WORLD);
01329           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01330                     1, MPI_COMM_WORLD);
01331         }
01332     }
01333   else
01334     {
01335       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01336                 MPI_COMM_WORLD, &mpi_stat);
```

```
01337       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01338                 MPI_COMM_WORLD, &mpi_stat);
01339     }
01340 #endif
01341 #if DEBUG_OPTIMIZE
01342   fprintf (stderr, "optimize_refine:  end\n");
01343 #endif
01344 }
01345
01349 static void
01350 optimize_step ()
01351 {
01352 #if DEBUG_OPTIMIZE
01353   fprintf (stderr, "optimize_step:  start\n");
01354 #endif
01355   optimize_algorithm ();
01356   if (optimize->nsteps)
01357     optimize_climbing ();
01358 #if DEBUG_OPTIMIZE
01359   fprintf (stderr, "optimize_step:  end\n");
01360 #endif
01361 }
01362
01366 static inline void
01367 optimize_iterate ()
01368 {
01369   unsigned int i;
01370 #if DEBUG_OPTIMIZE
01371   fprintf (stderr, "optimize_iterate:  start\n");
01372 #endif
01373   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01374   optimize->value_old =
01375     (double *) g_malloc (optimize->nbest * optimize->nvariables *
01376                          sizeof (double));
01377   optimize_step ();
01378   optimize_save_old ();
01379   optimize_refine ();
01380   optimize_print ();
01381   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01382     {
01383       optimize_step ();
01384       optimize_merge_old ();
01385       optimize_refine ();
01386       optimize_print ();
01387     }
01388 #if DEBUG_OPTIMIZE
01389   fprintf (stderr, "optimize_iterate:  end\n");
01390 #endif
01391 }
01392
01396 void
01397 optimize_free ()
01398 {
01399   unsigned int i, j;
01400 #if DEBUG_OPTIMIZE
01401   fprintf (stderr, "optimize_free:  start\n");
01402 #endif
01403   for (j = 0; j < optimize->ninputs; ++j)
01404     {
01405       for (i = 0; i < optimize->nexperiments; ++i)
01406         g_mapped_file_unref (optimize->file[j][i]);
01407       g_free (optimize->file[j]);
01408     }
01409   g_free (optimize->error_old);
01410   g_free (optimize->value_old);
01411   g_free (optimize->value);
01412   g_free (optimize->genetic_variable);
01413 #if DEBUG_OPTIMIZE
01414   fprintf (stderr, "optimize_free:  end\n");
01415 #endif
01416 }
01417
01421 void
01422 optimize_open ()
01423 {
01424   GTimeZone *tz;
01425   GDateTime *t0, *t;
01426   unsigned int i, j;
01427
01428 #if DEBUG_OPTIMIZE
01429   char *buffer;
01430   fprintf (stderr, "optimize_open:  start\n");
01431 #endif
01432
01433   // Getting initial time
01434 #if DEBUG_OPTIMIZE
01435   fprintf (stderr, "optimize_open:  getting initial time\n");
```

```
01436 #endif
01437   tz = g_time_zone_new_utc ();
01438   t0 = g_date_time_new_now (tz);
01439
01440   // Obtaining and initing the pseudo-random numbers generator seed
01441 #if DEBUG_OPTIMIZE
01442   fprintf (stderr, "optimize_open:  getting initial seed\n");
01443 #endif
01444   if (optimize->seed == DEFAULT_RANDOM_SEED)
01445     optimize->seed = input->seed;
01446   gsl_rng_set (optimize->rng, optimize->seed);
01447
01448   // Replacing the working directory
01449 #if DEBUG_OPTIMIZE
01450   fprintf (stderr, "optimize_open:  replacing the working directory\n");
01451 #endif
01452   g_chdir (input->directory);
01453
01454   // Getting results file names
01455   optimize->result = input->result;
01456   optimize->variables = input->variables;
01457
01458   // Obtaining the simulator file
01459   optimize->simulator = input->simulator;
01460
01461   // Obtaining the evaluator file
01462   optimize->evaluator = input->evaluator;
01463
01464   // Reading the algorithm
01465   optimize->algorithm = input->algorithm;
01466   switch (optimize->algorithm)
01467     {
01468     case ALGORITHM_MONTE_CARLO:
01469       optimize_algorithm = optimize_MonteCarlo;
01470       break;
01471     case ALGORITHM_SWEEP:
01472       optimize_algorithm = optimize_sweep;
01473       break;
01474     case ALGORITHM_ORTHOGONAL:
01475       optimize_algorithm = optimize_orthogonal;
01476       break;
01477     default:
01478       optimize_algorithm = optimize_genetic;
01479       optimize->mutation_ratio = input->mutation_ratio;
01480       optimize->reproduction_ratio = input->reproduction_ratio;
01481       optimize->adaptation_ratio = input->adaptation_ratio;
01482     }
01483   optimize->nvariables = input->nvariables;
01484   optimize->nsimulations = input->nsimulations;
01485   optimize->niterations = input->niterations;
01486   optimize->nbest = input->nbest;
01487   optimize->tolerance = input->tolerance;
01488   optimize->nsteps = input->nsteps;
01489   optimize->nestimates = 0;
01490   optimize->threshold = input->threshold;
01491   optimize->stop = 0;
01492   if (input->nsteps)
01493     {
01494       optimize->relaxation = input->relaxation;
01495       switch (input->climbing)
01496         {
01497         case CLIMBING_METHOD_COORDINATES:
01498           optimize->nestimates = 2 * optimize->nvariables;
01499           optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
01500           break;
01501         default:
01502           optimize->nestimates = input->nestimates;
01503           optimize_estimate_climbing = optimize_estimate_climbing_random;
01504         }
01505     }
01506
01507 #if DEBUG_OPTIMIZE
01508   fprintf (stderr, "optimize_open:  nbest=%u\n", optimize->nbest);
01509 #endif
01510   optimize->simulation_best
01511     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01512   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01513
01514   // Reading the experimental data
01515 #if DEBUG_OPTIMIZE
01516   buffer = g_get_current_dir ();
01517   fprintf (stderr, "optimize_open:  current directory=%s\n", buffer);
01518   g_free (buffer);
01519 #endif
01520   optimize->nexperiments = input->nexperiments;
01521   optimize->ninputs = input->experiment->ninputs;
01522   optimize->experiment
```

```
01523     = (char **) alloca (input->nexperiments * sizeof (char *));
01524   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01525   for (i = 0; i < input->experiment->ninputs; ++i)
01526     optimize->file[i] = (GMappedFile **)
01527       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01528   for (i = 0; i < input->nexperiments; ++i)
01529     {
01530 #if DEBUG_OPTIMIZE
01531       fprintf (stderr, "optimize_open:  i=%u\n", i);
01532 #endif
01533       optimize->experiment[i] = input->experiment[i].name;
01534       optimize->weight[i] = input->experiment[i].weight;
01535 #if DEBUG_OPTIMIZE
01536       fprintf (stderr, "optimize_open:  experiment=%s weight=%lg\n",
01537                optimize->experiment[i], optimize->weight[i]);
01538 #endif
01539       for (j = 0; j < input->experiment->ninputs; ++j)
01540         {
01541 #if DEBUG_OPTIMIZE
01542           fprintf (stderr, "optimize_open:  stencil%u\n", j + 1);
01543 #endif
01544           optimize->file[j][i]
01545             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01546         }
01547     }
01548
01549   // Reading the variables data
01550 #if DEBUG_OPTIMIZE
01551   fprintf (stderr, "optimize_open:  reading variables\n");
01552 #endif
01553   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01554   j = input->nvariables * sizeof (double);
01555   optimize->rangemin = (double *) alloca (j);
01556   optimize->rangeminabs = (double *) alloca (j);
01557   optimize->rangemax = (double *) alloca (j);
01558   optimize->rangemaxabs = (double *) alloca (j);
01559   optimize->step = (double *) alloca (j);
01560   j = input->nvariables * sizeof (unsigned int);
01561   optimize->precision = (unsigned int *) alloca (j);
01562   optimize->nsweeps = (unsigned int *) alloca (j);
01563   optimize->nbits = (unsigned int *) alloca (j);
01564   for (i = 0; i < input->nvariables; ++i)
01565     {
01566       optimize->label[i] = input->variable[i].name;
01567       optimize->rangemin[i] = input->variable[i].rangemin;
01568       optimize->rangeminabs[i] = input->variable[i].rangeminabs;
01569       optimize->rangemax[i] = input->variable[i].rangemax;
01570       optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
01571       optimize->precision[i] = input->variable[i].precision;
01572       optimize->step[i] = input->variable[i].step;
01573       optimize->nsweeps[i] = input->variable[i].nsweeps;
01574       optimize->nbits[i] = input->variable[i].nbits;
01575     }
01576   if (input->algorithm == ALGORITHM_SWEEP
01577       || input->algorithm == ALGORITHM_ORTHOGONAL)
01578     {
01579       optimize->nsimulations = 1;
01580       for (i = 0; i < input->nvariables; ++i)
01581         {
01582           optimize->nsimulations *= optimize->nsweeps[i];
01583 #if DEBUG_OPTIMIZE
01584           fprintf (stderr, "optimize_open:  nsweeps=%u nsimulations=%u\n",
01585                    optimize->nsweeps[i], optimize->nsimulations);
01586 #endif
01587         }
01588     }
01589   if (optimize->nsteps)
01590     optimize->climbing
01591       = (double *) alloca (optimize->nvariables * sizeof (double));
01592
01593   // Setting error norm
01594   switch (input->norm)
01595     {
01596     case ERROR_NORM_EUCLIDIAN:
01597       optimize_norm = optimize_norm_euclidian;
01598       break;
01599     case ERROR_NORM_MAXIMUM:
01600       optimize_norm = optimize_norm_maximum;
01601       break;
01602     case ERROR_NORM_P:
01603       optimize_norm = optimize_norm_p;
01604       optimize->p = input->p;
01605       break;
01606     default:
01607       optimize_norm = optimize_norm_taxicab;
01608     }
01609
```

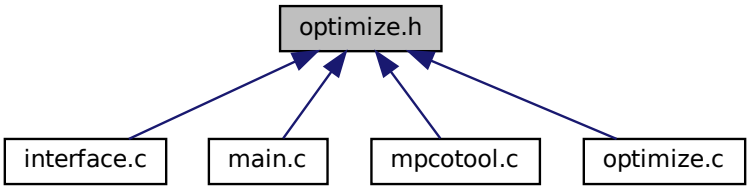
```
01610   // Allocating values
01611 #if DEBUG_OPTIMIZE
01612   fprintf (stderr, "optimize_open:  allocating variables\n");
01613   fprintf (stderr, "optimize_open:  nvariables=%u algorithm=%u\n",
01614            optimize->nvariables, optimize->algorithm);
01615 #endif
01616   optimize->genetic_variable = NULL;
01617   if (optimize->algorithm == ALGORITHM_GENETIC)
01618     {
01619       optimize->genetic_variable = (GeneticVariable *)
01620         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01621       for (i = 0; i < optimize->nvariables; ++i)
01622         {
01623 #if DEBUG_OPTIMIZE
01624           fprintf (stderr, "optimize_open:  i=%u min=%lg max=%lg nbits=%u\n",
01625                    i, optimize->rangemin[i], optimize->rangemax[i],
01626                    optimize->nbits[i]);
01627 #endif
01628           optimize->genetic_variable[i].minimum = optimize->rangemin[i];
01629           optimize->genetic_variable[i].maximum = optimize->rangemax[i];
01630           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01631         }
01632     }
01633 #if DEBUG_OPTIMIZE
01634   fprintf (stderr, "optimize_open:  nvariables=%u nsimulations=%u\n",
01635            optimize->nvariables, optimize->nsimulations);
01636 #endif
01637   optimize->value = (double *)
01638     g_malloc ((optimize->nsimulations
01639                + optimize->nestimates * optimize->nsteps)
01640               * optimize->nvariables * sizeof (double));
01641
01642   // Calculating simulations to perform for each task
01643 #if HAVE_MPI
01644 #if DEBUG_OPTIMIZE
01645   fprintf (stderr, "optimize_open:  rank=%u ntasks=%u\n",
01646            optimize->mpi_rank, ntasks);
01647 #endif
01648   optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
01649   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
01650   if (optimize->nsteps)
01651     {
01652       optimize->nstart_climbing
01653         = optimize->mpi_rank * optimize->nestimates / ntasks;
01654       optimize->nend_climbing
01655         = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
01656     }
01657 #else
01658   optimize->nstart = 0;
01659   optimize->nend = optimize->nsimulations;
01660   if (optimize->nsteps)
01661     {
01662       optimize->nstart_climbing = 0;
01663       optimize->nend_climbing = optimize->nestimates;
01664     }
01665 #endif
01666 #if DEBUG_OPTIMIZE
01667   fprintf (stderr, "optimize_open:  nstart=%u nend=%u\n", optimize->nstart,
01668            optimize->nend);
01669 #endif
01670
01671   // Calculating simulations to perform for each thread
01672   optimize->thread
01673     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01674   for (i = 0; i <= nthreads; ++i)
01675     {
01676       optimize->thread[i] = optimize->nstart
01677         + i * (optimize->nend - optimize->nstart) / nthreads;
01678 #if DEBUG_OPTIMIZE
01679       fprintf (stderr, "optimize_open:  i=%u thread=%u\n", i,
01680                optimize->thread[i]);
01681 #endif
01682     }
01683   if (optimize->nsteps)
01684     optimize->thread_climbing = (unsigned int *)
01685       alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01686
01687   // Opening result files
01688   optimize->file_result = g_fopen (optimize->result, "w");
01689   optimize->file_variables = g_fopen (optimize->variables, "w");
01690
01691   // Performing the algorithm
01692   switch (optimize->algorithm)
01693     {
01694       // Genetic algorithm
01695     case ALGORITHM_GENETIC:
01696       optimize_genetic ();
```

```
01697       break;
01698
01699       // Iterative algorithm
01700     default:
01701       optimize_iterate ();
01702     }
01703
01704   // Getting calculation time
01705   t = g_date_time_new_now (tz);
01706   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01707   g_date_time_unref (t);
01708   g_date_time_unref (t0);
01709   g_time_zone_unref (tz);
01710   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01711   fprintf (optimize->file_result, "%s = %.6lg s\n",
01712            _("Calculation time"), optimize->calculation_time);
01713
01714   // Closing result files
01715   fclose (optimize->file_variables);
01716   fclose (optimize->file_result);
01717
01718 #if DEBUG_OPTIMIZE
01719   fprintf (stderr, "optimize_open:  end\n");
01720 #endif
01721 }
```

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 1397 of file optimize.c.

```
01398 {
01399   unsigned int i, j;
01400 #if DEBUG_OPTIMIZE
01401   fprintf (stderr, "optimize_free:  start\n");
01402 #endif
01403   for (j = 0; j < optimize->ninputs; ++j)
01404     {
01405       for (i = 0; i < optimize->nexperiments; ++i)
01406         g_mapped_file_unref (optimize->file[j][i]);
01407       g_free (optimize->file[j]);
01408     }
01409   g_free (optimize->error_old);
01410   g_free (optimize->value_old);
01411   g_free (optimize->value);
01412   g_free (optimize->genetic_variable);
01413 #if DEBUG_OPTIMIZE
01414   fprintf (stderr, "optimize_free:  end\n");
01415 #endif
01416 }
```

4.23.2.2 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1422 of file optimize.c.

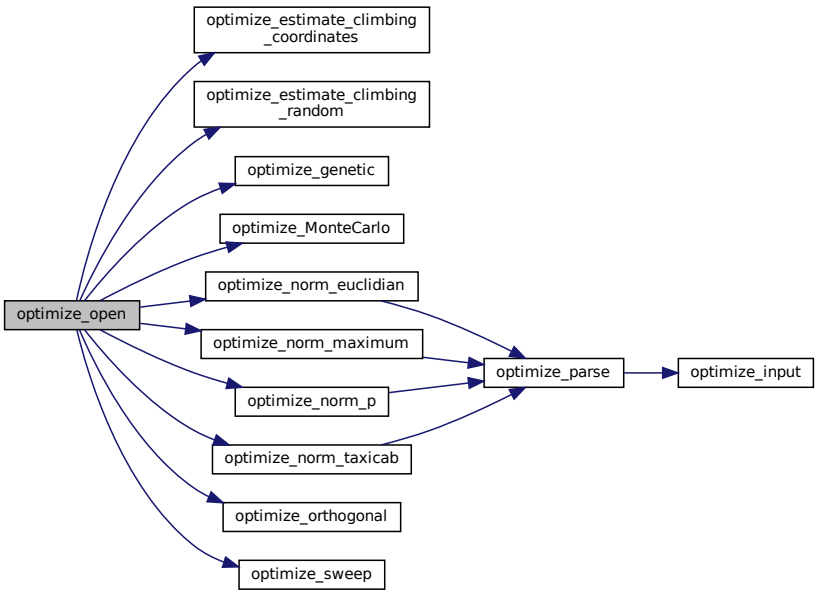
```
01423 {
01424   GTimeZone *tz;
01425   GDateTime *t0, *t;
01426   unsigned int i, j;
01427
01428 #if DEBUG_OPTIMIZE
01429   char *buffer;
01430   fprintf (stderr, "optimize_open:  start\n");
01431 #endif
01432
01433   // Getting initial time
01434 #if DEBUG_OPTIMIZE
01435   fprintf (stderr, "optimize_open:  getting initial time\n");
01436 #endif
01437   tz = g_time_zone_new_utc ();
01438   t0 = g_date_time_new_now (tz);
01439
01440   // Obtaining and initing the pseudo-random numbers generator seed
01441 #if DEBUG_OPTIMIZE
01442   fprintf (stderr, "optimize_open:  getting initial seed\n");
01443 #endif
01444   if (optimize->seed == DEFAULT_RANDOM_SEED)
01445     optimize->seed = input->seed;
01446   gsl_rng_set (optimize->rng, optimize->seed);
01447
01448   // Replacing the working directory
01449 #if DEBUG_OPTIMIZE
01450   fprintf (stderr, "optimize_open:  replacing the working directory\n");
01451 #endif
01452   g_chdir (input->directory);
01453
01454   // Getting results file names
01455   optimize->result = input->result;
01456   optimize->variables = input->variables;
01457
01458   // Obtaining the simulator file
01459   optimize->simulator = input->simulator;
01460
01461   // Obtaining the evaluator file
01462   optimize->evaluator = input->evaluator;
01463
01464   // Reading the algorithm
01465   optimize->algorithm = input->algorithm;
01466   switch (optimize->algorithm)
01467     {
01468     case ALGORITHM_MONTE_CARLO:
01469       optimize_algorithm = optimize_MonteCarlo;
01470       break;
01471     case ALGORITHM_SWEEP:
01472       optimize_algorithm = optimize_sweep;
01473       break;
01474     case ALGORITHM_ORTHOGONAL:
01475       optimize_algorithm = optimize_orthogonal;
01476       break;
01477     default:
01478       optimize_algorithm = optimize_genetic;
01479       optimize->mutation_ratio = input->mutation_ratio;
01480       optimize->reproduction_ratio = input->reproduction_ratio;
01481       optimize->adaptation_ratio = input->adaptation_ratio;
01482     }
01483   optimize->nvariables = input->nvariables;
01484   optimize->nsimulations = input->nsimulations;
01485   optimize->niterations = input->niterations;
01486   optimize->nbest = input->nbest;
01487   optimize->tolerance = input->tolerance;
01488   optimize->nsteps = input->nsteps;
01489   optimize->nestimates = 0;
01490   optimize->threshold = input->threshold;
01491   optimize->stop = 0;
01492   if (input->nsteps)
01493     {
01494       optimize->relaxation = input->relaxation;
01495       switch (input->climbing)
01496         {
01497         case CLIMBING_METHOD_COORDINATES:
01498           optimize->nestimates = 2 * optimize->nvariables;
```

```
01499           optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
01500           break;
01501         default:
01502           optimize->nestimates = input->nestimates;
01503           optimize_estimate_climbing = optimize_estimate_climbing_random;
01504         }
01505     }
01506
01507 #if DEBUG_OPTIMIZE
01508   fprintf (stderr, "optimize_open:  nbest=%u\n", optimize->nbest);
01509 #endif
01510   optimize->simulation_best
01511     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01512   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01513
01514   // Reading the experimental data
01515 #if DEBUG_OPTIMIZE
01516   buffer = g_get_current_dir ();
01517   fprintf (stderr, "optimize_open:  current directory=%s\n", buffer);
01518   g_free (buffer);
01519 #endif
01520   optimize->nexperiments = input->nexperiments;
01521   optimize->ninputs = input->experiment->ninputs;
01522   optimize->experiment
01523     = (char **) alloca (input->nexperiments * sizeof (char *));
01524   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01525   for (i = 0; i < input->experiment->ninputs; ++i)
01526     optimize->file[i] = (GMappedFile **)
01527       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01528   for (i = 0; i < input->nexperiments; ++i)
01529     {
01530 #if DEBUG_OPTIMIZE
01531       fprintf (stderr, "optimize_open:  i=%u\n", i);
01532 #endif
01533       optimize->experiment[i] = input->experiment[i].name;
01534       optimize->weight[i] = input->experiment[i].weight;
01535 #if DEBUG_OPTIMIZE
01536       fprintf (stderr, "optimize_open:  experiment=%s weight=%lg\n",
01537                optimize->experiment[i], optimize->weight[i]);
01538 #endif
01539       for (j = 0; j < input->experiment->ninputs; ++j)
01540         {
01541 #if DEBUG_OPTIMIZE
01542           fprintf (stderr, "optimize_open:  stencil%u\n", j + 1);
01543 #endif
01544           optimize->file[j][i]
01545             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01546         }
01547     }
01548
01549   // Reading the variables data
01550 #if DEBUG_OPTIMIZE
01551   fprintf (stderr, "optimize_open:  reading variables\n");
01552 #endif
01553   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01554   j = input->nvariables * sizeof (double);
01555   optimize->rangemin = (double *) alloca (j);
01556   optimize->rangeminabs = (double *) alloca (j);
01557   optimize->rangemax = (double *) alloca (j);
01558   optimize->rangemaxabs = (double *) alloca (j);
01559   optimize->step = (double *) alloca (j);
01560   j = input->nvariables * sizeof (unsigned int);
01561   optimize->precision = (unsigned int *) alloca (j);
01562   optimize->nsweeps = (unsigned int *) alloca (j);
01563   optimize->nbits = (unsigned int *) alloca (j);
01564   for (i = 0; i < input->nvariables; ++i)
01565     {
01566       optimize->label[i] = input->variable[i].name;
01567       optimize->rangemin[i] = input->variable[i].rangemin;
01568       optimize->rangeminabs[i] = input->variable[i].rangeminabs;
01569       optimize->rangemax[i] = input->variable[i].rangemax;
01570       optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
01571       optimize->precision[i] = input->variable[i].precision;
01572       optimize->step[i] = input->variable[i].step;
01573       optimize->nsweeps[i] = input->variable[i].nsweeps;
01574       optimize->nbits[i] = input->variable[i].nbits;
01575     }
01576   if (input->algorithm == ALGORITHM_SWEEP
01577       || input->algorithm == ALGORITHM_ORTHOGONAL)
01578     {
01579       optimize->nsimulations = 1;
01580       for (i = 0; i < input->nvariables; ++i)
01581         {
01582           optimize->nsimulations *= optimize->nsweeps[i];
01583 #if DEBUG_OPTIMIZE
01584           fprintf (stderr, "optimize_open:  nsweeps=%u nsimulations=%u\n",
01585                    optimize->nsweeps[i], optimize->nsimulations);
```

```
01586 #endif
01587         }
01588     }
01589   if (optimize->nsteps)
01590     optimize->climbing
01591       = (double *) alloca (optimize->nvariables * sizeof (double));
01592
01593   // Setting error norm
01594   switch (input->norm)
01595     {
01596     case ERROR_NORM_EUCLIDIAN:
01597       optimize_norm = optimize_norm_euclidian;
01598       break;
01599     case ERROR_NORM_MAXIMUM:
01600       optimize_norm = optimize_norm_maximum;
01601       break;
01602     case ERROR_NORM_P:
01603       optimize_norm = optimize_norm_p;
01604       optimize->p = input->p;
01605       break;
01606     default:
01607       optimize_norm = optimize_norm_taxicab;
01608     }
01609
01610   // Allocating values
01611 #if DEBUG_OPTIMIZE
01612   fprintf (stderr, "optimize_open:  allocating variables\n");
01613   fprintf (stderr, "optimize_open:  nvariables=%u algorithm=%u\n",
01614            optimize->nvariables, optimize->algorithm);
01615 #endif
01616   optimize->genetic_variable = NULL;
01617   if (optimize->algorithm == ALGORITHM_GENETIC)
01618     {
01619       optimize->genetic_variable = (GeneticVariable *)
01620         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01621       for (i = 0; i < optimize->nvariables; ++i)
01622         {
01623 #if DEBUG_OPTIMIZE
01624           fprintf (stderr, "optimize_open:  i=%u min=%lg max=%lg nbits=%u\n",
01625                    i, optimize->rangemin[i], optimize->rangemax[i],
01626                    optimize->nbits[i]);
01627 #endif
01628           optimize->genetic_variable[i].minimum = optimize->rangemin[i];
01629           optimize->genetic_variable[i].maximum = optimize->rangemax[i];
01630           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01631         }
01632     }
01633 #if DEBUG_OPTIMIZE
01634   fprintf (stderr, "optimize_open:  nvariables=%u nsimulations=%u\n",
01635            optimize->nvariables, optimize->nsimulations);
01636 #endif
01637   optimize->value = (double *)
01638     g_malloc ((optimize->nsimulations
01639                + optimize->nestimates * optimize->nsteps)
01640               * optimize->nvariables * sizeof (double));
01641
01642   // Calculating simulations to perform for each task
01643 #if HAVE_MPI
01644 #if DEBUG_OPTIMIZE
01645   fprintf (stderr, "optimize_open:  rank=%u ntasks=%u\n",
01646            optimize->mpi_rank, ntasks);
01647 #endif
01648   optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
01649   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
01650   if (optimize->nsteps)
01651     {
01652       optimize->nstart_climbing
01653         = optimize->mpi_rank * optimize->nestimates / ntasks;
01654       optimize->nend_climbing
01655         = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
01656     }
01657 #else
01658   optimize->nstart = 0;
01659   optimize->nend = optimize->nsimulations;
01660   if (optimize->nsteps)
01661     {
01662       optimize->nstart_climbing = 0;
01663       optimize->nend_climbing = optimize->nestimates;
01664     }
01665 #endif
01666 #if DEBUG_OPTIMIZE
01667   fprintf (stderr, "optimize_open:  nstart=%u nend=%u\n", optimize->nstart,
01668            optimize->nend);
01669 #endif
01670
01671   // Calculating simulations to perform for each thread
01672   optimize->thread
```

```
01673     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01674   for (i = 0; i <= nthreads; ++i)
01675     {
01676       optimize->thread[i] = optimize->nstart
01677         + i * (optimize->nend - optimize->nstart) / nthreads;
01678 #if DEBUG_OPTIMIZE
01679       fprintf (stderr, "optimize_open:  i=%u thread=%u\n", i,
01680                optimize->thread[i]);
01681 #endif
01682     }
01683   if (optimize->nsteps)
01684     optimize->thread_climbing = (unsigned int *)
01685       alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01686
01687   // Opening result files
01688   optimize->file_result = g_fopen (optimize->result, "w");
01689   optimize->file_variables = g_fopen (optimize->variables, "w");
01690
01691   // Performing the algorithm
01692   switch (optimize->algorithm)
01693     {
01694       // Genetic algorithm
01695     case ALGORITHM_GENETIC:
01696       optimize_genetic ();
01697       break;
01698
01699       // Iterative algorithm
01700     default:
01701       optimize_iterate ();
01702     }
01703
01704   // Getting calculation time
01705   t = g_date_time_new_now (tz);
01706   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01707   g_date_time_unref (t);
01708   g_date_time_unref (t0);
01709   g_time_zone_unref (tz);
01710   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01711   fprintf (optimize->file_result, "%s = %.6lg s\n",
01712            _("Calculation time"), optimize->calculation_time);
01713
01714   // Closing result files
01715   fclose (optimize->file_variables);
01716   fclose (optimize->file_result);
01717
01718 #if DEBUG_OPTIMIZE
01719   fprintf (stderr, "optimize_open:  end\n");
01720 #endif
01721 }
```

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 80 of file optimize.c.

4.23.3.2 optimize

```
Optimize optimize[1]  [extern]
```

Optimization data.

Definition at line 79 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
00038 #ifndef OPTIMIZE__H
00039 #define OPTIMIZE__H 1
00040
00045 typedef struct
00046 {
00047   GMappedFile **file[MAX_NINPUTS];
00048   char **experiment;
00049   char **label;
00050   gsl_rng *rng;
```

```
00051   GeneticVariable *genetic_variable;
00053   FILE *file_result;
00054   FILE *file_variables;
00055   char *result;
00056   char *variables;
00057   char *simulator;
00058   char *evaluator;
00060   double *value;
00061   double *rangemin;
00062   double *rangemax;
00063   double *rangeminabs;
00064   double *rangemaxabs;
00065   double *error_best;
00066   double *weight;
00067   double *step;
00068   double *climbing;
00069   double *value_old;
00071   double *error_old;
00073   unsigned int *precision;
00074   unsigned int *nsweeps;
00075   unsigned int *nbits;
00077   unsigned int *thread;
00079   unsigned int *thread_climbing;
00082   unsigned int *simulation_best;
00083   double tolerance;
00084   double mutation_ratio;
00085   double reproduction_ratio;
00086   double adaptation_ratio;
00087   double relaxation;
00088   double calculation_time;
00089   double p;
00090   double threshold;
00091   unsigned long int seed;
00093   unsigned int nvariables;
00094   unsigned int nexperiments;
00095   unsigned int ninputs;
00096   unsigned int nsimulations;
00097   unsigned int nsteps;
00099   unsigned int nestimates;
00101   unsigned int algorithm;
00102   unsigned int nstart;
00103   unsigned int nend;
00104   unsigned int nstart_climbing;
00106   unsigned int nend_climbing;
00108   unsigned int niterations;
00109   unsigned int nbest;
00110   unsigned int nsaveds;
00111   unsigned int stop;
00112 #if HAVE_MPI
00113   int mpi_rank;
00114 #endif
00115 } Optimize;
00116
00121 typedef struct
00122 {
00123   unsigned int thread;
00124 } ParallelData;
00125
00126 // Global variables
00127 extern int ntasks;
00128 extern unsigned int nthreads;
00129 extern unsigned int nthreads_climbing;
00130 extern GMutex mutex[1];
00131 extern Optimize optimize[1];
00132
00133 // Public functions
00134 void optimize_free ();
00135 void optimize_open ();
00136
00137 #endif
```

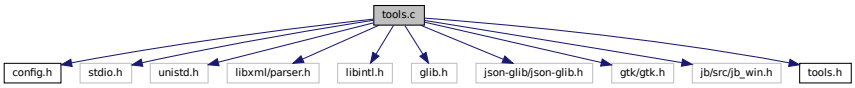
4.25 tools.c File Reference

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:



Functions

- void process_pending ()
- unsigned int gtk_array_get_active (GtkRadioButton *array[], unsigned int n)

Variables

- GtkWindow * main_window

  Main GtkWindow.
- 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

Definition in file tools.c.

4.25.2 Function Documentation

4.25.2.1 gtk_array_get_active()

```
unsigned int gtk_array_get_active (
            GtkRadioButton * array[],
            unsigned int n )
```

Function to get the active GtkRadioButton.

Returns

Active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Definition at line 82 of file tools.c.

```
00089 {
00090   unsigned int i;
00091   for (i = 0; i < n; ++i)
00092     if (gtk_check_button_get_active (array[i]))
00093       break;
00094   return i;
00095 }
```

4.25.2.2 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 69 of file tools.c.

```
00070 {
00071   GMainContext *context = g_main_context_default ();
00072   while (g_main_context_pending (context))
00073     g_main_context_iteration (context, 0);
00074 }
```

4.25.3 Variable Documentation

4.25.3.1 error_message

```
char* error_message
```

Error message.

Definition at line 59 of file tools.c.

4.25.3.2 main_window

```
GtkWindow* main_window
```

Main GtkWindow.

Definition at line 56 of file tools.c.

4.25.3.3 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.

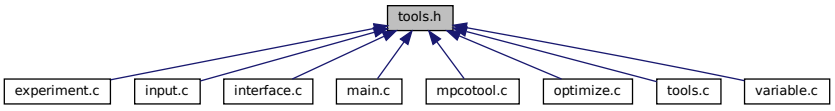
```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool.  A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and 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
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <unistd.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #include <glib.h>
00045 #include <json-glib/json-glib.h>
00046 #ifdef G_OS_WIN32
00047 #include <windows.h>
00048 #endif
00049 #if HAVE_GTK
00050 #include <gtk/gtk.h>
00051 #endif
00052 #include "jb/src/jb_win.h"
00053 #include "tools.h"
00054
00055 #if HAVE_GTK
00056 GtkWindow *main_window;
00057 #endif
00058
00059 char *error_message;
00060 void (*show_pending) () = NULL;
00062
00063 #if HAVE_GTK
00064
00068 void
00069 process_pending ()
00070 {
00071   GMainContext *context = g_main_context_default ();
00072   while (g_main_context_pending (context))
00073     g_main_context_iteration (context, 0);
00074 }
00075
00081 unsigned int
```

```
00082 gtk_array_get_active (
00083 #if !GTK4
00084                       GtkRadioButton * array[],
00085 #else
00086                       GtkCheckButton * array[],
00087 #endif
00088                       unsigned int n)
00089 {
00090   unsigned int i;
00091   for (i = 0; i < n; ++i)
00092     if (gtk_check_button_get_active (array[i]))
00093       break;
00094   return i;
00095 }
00096
00097 #endif
```

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.
- #define G_APPLICATION_DEFAULT_FLAGS G_APPLICATION_FLAGS_NONE

Functions

- void process_pending ()
- unsigned int gtk_array_get_active (GtkRadioButton *array[], unsigned int n)

Variables

- GtkWindow * main_window

  Main GtkWindow.
- 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

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 G_APPLICATION_DEFAULT_FLAGS

```
#define G_APPLICATION_DEFAULT_FLAGS G_APPLICATION_FLAGS_NONE
```

Definition at line 64 of file tools.h.

4.27.2.3 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 Function Documentation

4.27.3.1 gtk_array_get_active()

```
unsigned int gtk_array_get_active (
            GtkRadioButton * array[],
            unsigned int n )
```

Function to get the active GtkRadioButton.

Returns

Active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Definition at line 82 of file tools.c.

```
00089 {
00090   unsigned int i;
00091   for (i = 0; i < n; ++i)
00092     if (gtk_check_button_get_active (array[i]))
00093       break;
00094   return i;
00095 }
```

4.27.3.2 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 69 of file tools.c.

```
00070 {
00071   GMainContext *context = g_main_context_default ();
00072   while (g_main_context_pending (context))
00073     g_main_context_iteration (context, 0);
00074 }
```

4.27.4 Variable Documentation

4.27.4.1 error_message

```
char* error_message  [extern]
```

Error message.

Definition at line 59 of file tools.c.

4.27.4.2 main_window

```
GtkWindow* main_window  [extern]
```

Main GtkWindow.

Definition at line 56 of file tools.c.

4.27.4.3 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.

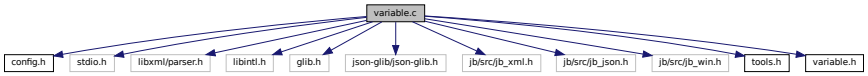
```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool.  A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2023, AUTHORS.
00009
00010 Redistribution and 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
00038 #ifndef TOOLS__H
00039 #define TOOLS__H 1
00040
00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWindow *main_window;
00051 #else
00052 #define ERROR_TYPE 0
00053 #define INFO_TYPE 0
00054 #endif
00055
00056 extern char *error_message;
00057 extern void (*show_pending) ();
00058
00059 // Public functions
00060
00061 #if HAVE_GTK
00062
00063 #ifndef G_APPLICATION_DEFAULT_FLAGS
00064 #define G_APPLICATION_DEFAULT_FLAGS G_APPLICATION_FLAGS_NONE
00065 #endif
00066
00067 void process_pending ();
00068
00069 #if !GTK4
00070 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00071 #else
00072 unsigned int gtk_array_get_active (GtkCheckButton * array[], unsigned int n);
00073 #endif
00074
00075 #endif
00076
00077 #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

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

variable	Variable struct.
message	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

variable	Variable struct.
type	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

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

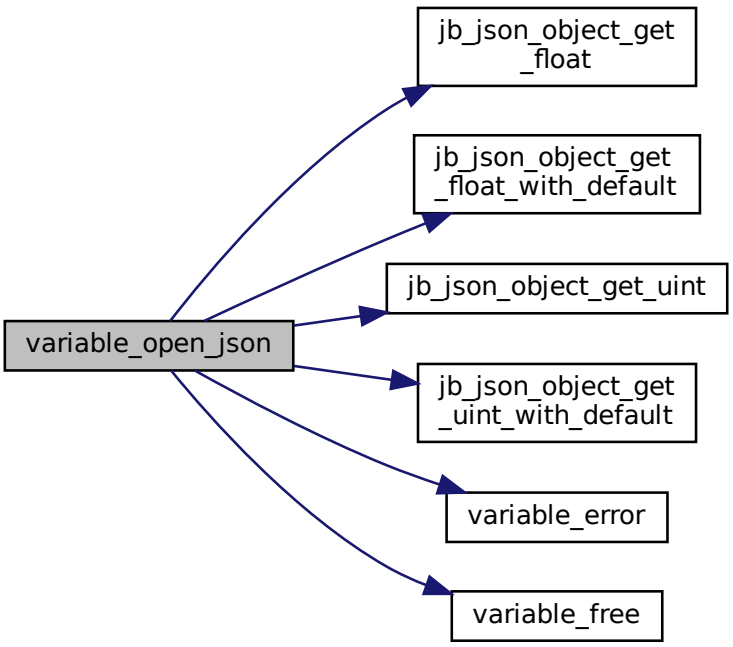
Definition at line 270 of file variable.c.

```
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       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

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

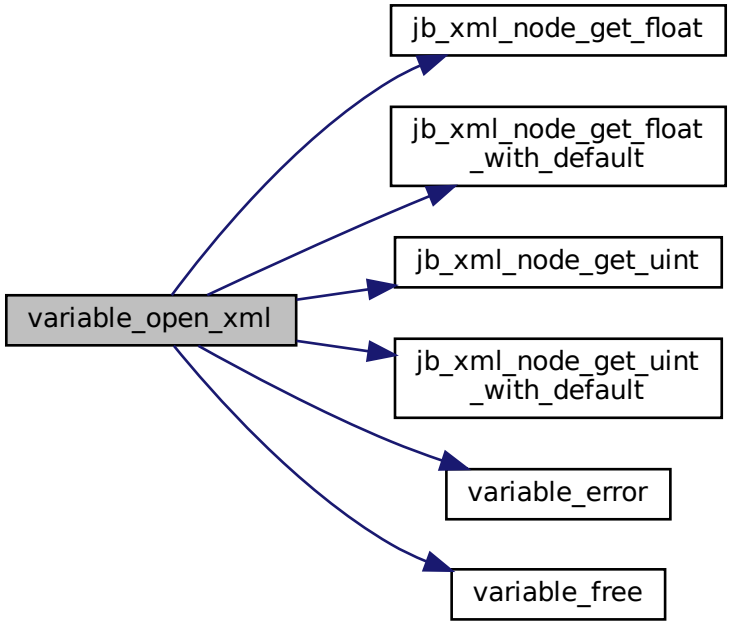
Definition at line 107 of file variable.c.

```
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
00139       if (!error_code)
00140         {
00141           variable_error (variable, _("bad absolute minimum"));
00142           goto exit_on_error;
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
00252 #if DEBUG_VARIABLE
00253   fprintf (stderr, "variable_open_xml:  end\n");
00254 #endif
00255   return 1;
00256 exit_on_error:
00257   variable_free (variable, INPUT_TYPE_XML);
00258 #if DEBUG_VARIABLE
00259   fprintf (stderr, "variable_open_xml:  end\n");
00260 #endif
00261   return 0;
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:

```
= {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
}
```

Array of C-strings with variable formats.

Definition at line 53 of file variable.c.

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.

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
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "jb/src/jb_xml.h"
00046 #include "jb/src/jb_json.h"
00047 #include "jb/src/jb_win.h"
00048 #include "tools.h"
00049 #include "variable.h"
00050
00051 #define DEBUG_VARIABLE 0
00052
00053 const char *format[NPRECISIONS] = {
00054   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00055   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00056 };
00057
00058 const double precision[NPRECISIONS] = {
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 };
00062
00066 void
00067 variable_free (Variable * variable,
00069                unsigned int type)
```

```
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 }
00083
00087 void
00088 variable_error (Variable * variable,
00090                 char *message)
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 }
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
00139       if (!error_code)
00140         {
00141           variable_error (variable, _("bad absolute minimum"));
00142           goto exit_on_error;
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
00252 #if DEBUG_VARIABLE
00253   fprintf (stderr, "variable_open_xml:  end\n");
00254 #endif
00255   return 1;
```

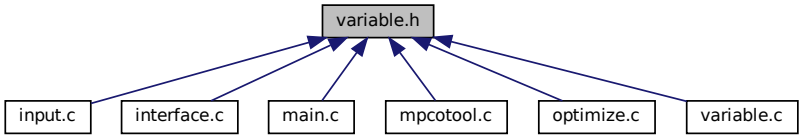
```
00256 exit_on_error:
00257   variable_free (variable, INPUT_TYPE_XML);
00258 #if DEBUG_VARIABLE
00259   fprintf (stderr, "variable_open_xml:  end\n");
00260 #endif
00261   return 0;
00262 }
00263
00269 int
00270 variable_open_json (Variable * variable,
00271                     JsonNode * node,
00272                     unsigned int algorithm,
00273                     unsigned int nsteps)
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       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, 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.

4.31.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Definition in file variable.h.

4.31.2 Enumeration Type Documentation

4.31.2.1 Algorithm

```
enum Algorithm
```

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO	Monte-Carlo algorithm.
ALGORITHM_SWEEP	Sweep algorithm.
ALGORITHM_GENETIC	Genetic algorithm.
ALGORITHM_ORTHOGONAL	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

variable	Variable struct.
message	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

variable	Variable struct.
type	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,
            JsonNode * node,
            unsigned int algorithm,
            unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

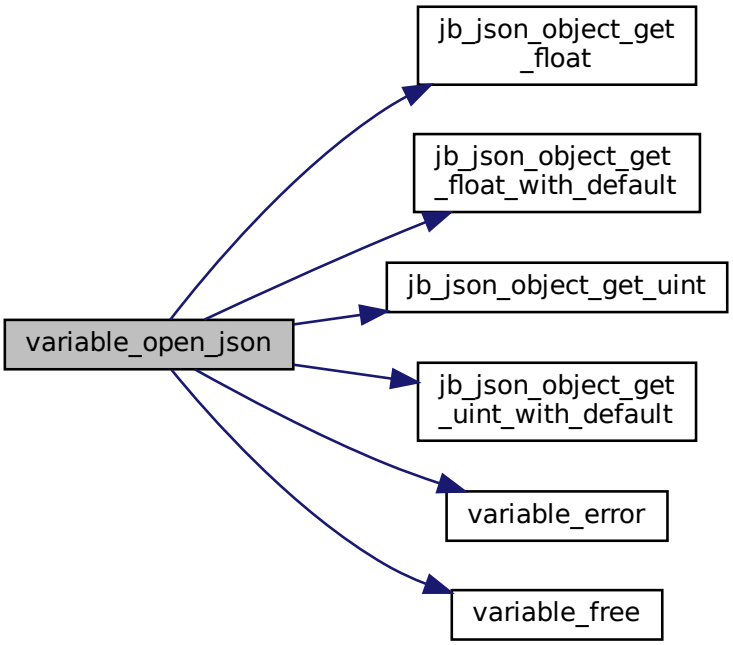
Definition at line 270 of file variable.c.

```
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       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:



4.31.3.4 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

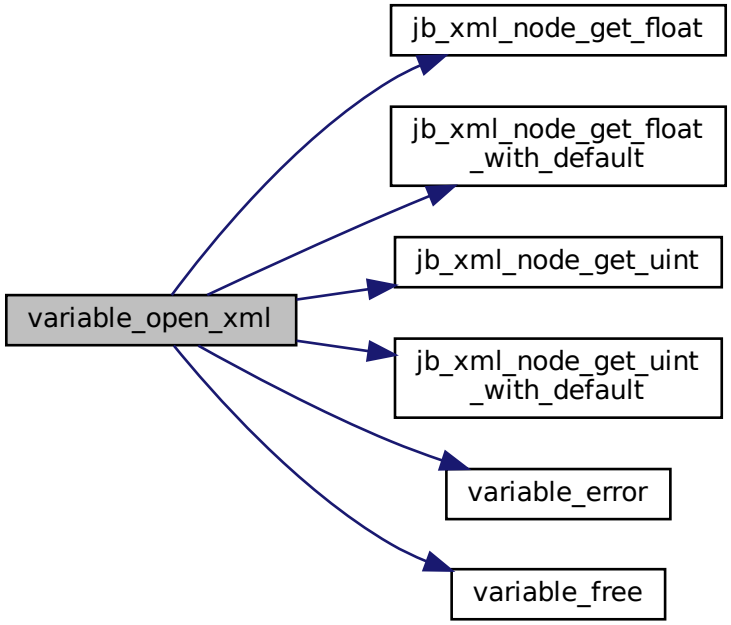
variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

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
00139       if (!error_code)
00140         {
00141           variable_error (variable, _("bad absolute minimum"));
00142           goto exit_on_error;
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
00252 #if DEBUG_VARIABLE
00253   fprintf (stderr, "variable_open_xml:  end\n");
00254 #endif
00255   return 1;
00256 exit_on_error:
00257   variable_free (variable, INPUT_TYPE_XML);
00258 #if DEBUG_VARIABLE
00259   fprintf (stderr, "variable_open_xml:  end\n");
00260 #endif
00261   return 0;
00262 }
```

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.

4.31.4.2 precision

```
const double precision[NPRECISIONS]  [extern]
```

Array of variable precisions.

Definition at line 58 of file variable.c.

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
00038 #ifndef VARIABLE__H
00039 #define VARIABLE__H 1
00040
00042 enum Algorithm
00043 {
00044   ALGORITHM_MONTE_CARLO = 0,
00045   ALGORITHM_SWEEP = 1,
00046   ALGORITHM_GENETIC = 2,
00047   ALGORITHM_ORTHOGONAL = 3
00048 };
00049
00054 typedef struct
00055 {
00056   char *name;
00057   double rangemin;
00058   double rangemax;
00059   double rangeminabs;
00060   double rangemaxabs;
00061   double step;
00062   unsigned int precision;
00063   unsigned int nsweeps;
00064   unsigned int nbits;
00065 } Variable;
00066
00067 extern const char *format[NPRECISIONS];
00068 extern const double precision[NPRECISIONS];
00069
00070 // Public functions
00071 void variable_free (Variable * variable, unsigned int type);
00072 void variable_error (Variable * variable, char *message);
00073 int variable_open_xml (Variable * variable, xmlNode * node,
00074                        unsigned int algorithm, unsigned int nsteps);
00075 int variable_open_json (Variable * variable, JsonNode * node,
00076                         unsigned int algorithm, unsigned int nsteps);
00077
00078 #endif
```

Index

adaptation_ratio
    Input, 8
    Optimize, 16
Algorithm
    variable.h, 326
algorithm
    Input, 8
    Optimize, 16
ALGORITHM_GENETIC
    variable.h, 326
ALGORITHM_MONTE_CARLO
    variable.h, 326
ALGORITHM_ORTHOGONAL
    variable.h, 326
ALGORITHM_SWEEP
    variable.h, 326
application_directory
    Window, 39

box_buttons
    Window, 39
button_about
    Window, 39
button_add_experiment
    Window, 40
button_add_variable
    Window, 40
button_algorithm
    Window, 40
button_climbing
    Window, 40
button_evaluator
    Window, 40
button_exit
    Window, 41
button_experiment
    Window, 41
button_help
    Window, 41
button_norm
    Window, 41
button_open
    Window, 41
button_options
    Window, 42
button_remove_experiment
    Window, 42
button_remove_variable
    Window, 42
button_run
    Window, 42
button_save
    Window, 42
button_simulator
    Window, 43
button_template
    Window, 43

calculation_time
    Optimize, 17
check_climbing
    Window, 43
check_evaluator
    Window, 43
check_maxabs
    Window, 43
check_minabs
    Window, 44
check_template
    Window, 44
climbing
    Input, 8
    Optimize, 17
CLIMBING_METHOD_COORDINATES
    input.h, 129
CLIMBING_METHOD_RANDOM
    input.h, 129
ClimbingMethod
    input.h, 128
combo_experiment
    Window, 44
combo_variable
    Window, 44
config.h, 63
    DEFAULT_PRECISION, 66
    DEFAULT_RANDOM_SEED, 66
    DEFAULT_RELAXATION, 66
    HAVE_MPI, 67
    INPUT_TYPE, 79
    INPUT_TYPE_JSON, 79
    INPUT_TYPE_XML, 79
    LABEL_ABSOLUTE_MAXIMUM, 67
    LABEL_ABSOLUTE_MINIMUM, 67
    LABEL_ADAPTATION, 67
    LABEL_ALGORITHM, 67
    LABEL_CLIMBING, 68
    LABEL_COORDINATES, 68
    LABEL_EUCLIDIAN, 68
    LABEL_EVALUATOR, 68
    LABEL_EXPERIMENT, 68

LABEL_EXPERIMENTS, 69
LABEL_GENETIC, 69
LABEL_MAXIMUM, 69
LABEL_MINIMUM, 69
LABEL_MONTE_CARLO, 69
LABEL_MUTATION, 70
LABEL_NAME, 70
LABEL_NBEST, 70
LABEL_NBITS, 70
LABEL_NESTIMATES, 70
LABEL_NGENERATIONS, 71
LABEL_NITERATIONS, 71
LABEL_NORM, 71
LABEL_NPOPULATION, 71
LABEL_NSIMULATIONS, 71
LABEL_NSTEPS, 72
LABEL_NSWEEPS, 72
LABEL_OPTIMIZE, 72
LABEL_ORTHOGONAL, 72
LABEL_P, 72
LABEL_PRECISION, 73
LABEL_RANDOM, 73
LABEL_RELAXATION, 73
LABEL_REPRODUCTION, 73
LABEL_RESULT_FILE, 73
LABEL_SEED, 74
LABEL_SIMULATOR, 74
LABEL_STEP, 74
LABEL_SWEEP, 74
LABEL_TAXICAB, 74
LABEL_TEMPLATE1, 75
LABEL_TEMPLATE2, 75
LABEL_TEMPLATE3, 75
LABEL_TEMPLATE4, 75
LABEL_TEMPLATE5, 75
LABEL_TEMPLATE6, 76
LABEL_TEMPLATE7, 76
LABEL_TEMPLATE8, 76
LABEL_THRESHOLD, 76
LABEL_TOLERANCE, 76
LABEL_VARIABLE, 77
LABEL_VARIABLES, 77
LABEL_VARIABLES_FILE, 77
LABEL_WEIGHT, 77
LOCALE_DIR, 77
MAX_NINPUTS, 78
NALGORITHMS, 78
NCLIMBINGS, 78
NNORMS, 78
NPRECISIONS, 78
PROGRAM_INTERFACE, 79

DEBUG_EXPERIMENT
experiment.c, 82
DEBUG_INPUT
input.c, 100
DEBUG_INTERFACE
interface.c, 136
DEBUG_MPCOTOOL
mpcotool.c, 238
DEBUG_OPTIMIZE
optimize.c, 248
DEBUG_VARIABLE
variable.c, 313
DEFAULT_PRECISION
config.h, 66
DEFAULT_RANDOM_SEED
config.h, 66
DEFAULT_RELAXATION
config.h, 66
dialog
Options, 28
Running, 31
dialog_evaluator
interface.c, 137
dialog_evaluator_close
interface.c, 137
dialog_name_experiment_close
interface.c, 138
dialog_open_close
interface.c, 138
dialog_options_close
interface.c, 140
dialog_save_close
interface.c, 140
dialog_simulator
interface.c, 142
dialog_simulator_close
interface.c, 143
directory
Input, 8

entry_result
Window, 44
entry_variable
Window, 45
entry_variables
Window, 45
error_best
Optimize, 17
error_message
tools.c, 306
tools.h, 310
ERROR_NORM_EUCLIDIAN
input.h, 129
ERROR_NORM_MAXIMUM
input.h, 129
ERROR_NORM_P
input.h, 129
ERROR_NORM_TAXICAB
input.h, 129
error_old
Optimize, 17
ERROR_TYPE
tools.h, 309
ErrorNorm
input.h, 129
evaluator

Input, 9
Optimize, 17
Experiment, 5
name, 5
ninputs, 5
stencil, 6
weight, 6
experiment
Input, 9
Optimize, 18
Window, 45
experiment.c, 81
DEBUG_EXPERIMENT, 82
experiment_error, 82
experiment_free, 83
experiment_new, 83
experiment_open_json, 84
experiment_open_xml, 86
stencil, 88
experiment.h, 92
experiment_error, 93
experiment_free, 93
experiment_open_json, 94
experiment_open_xml, 96
stencil, 98
experiment_error
experiment.c, 82
experiment.h, 93
experiment_free
experiment.c, 83
experiment.h, 93
experiment_new
experiment.c, 83
experiment_open_json
experiment.c, 84
experiment.h, 94
experiment_open_xml
experiment.c, 86
experiment.h, 96

file
Optimize, 18
file_result
Optimize, 18
file_variables
Optimize, 18
format
variable.c, 319
variable.h, 333
frame_algorithm
Window, 45
frame_experiment
Window, 45
frame_norm
Window, 46
frame_variable
Window, 46

G_APPLICATION_DEFAULT_FLAGS
tools.h, 309
genetic_variable
Optimize, 18
grid
Options, 28
Running, 31
Window, 46
grid_algorithm
Window, 46
grid_climbing
Window, 46
grid_experiment
Window, 47
grid_files
Window, 47
grid_norm
Window, 47
grid_variable
Window, 47
gtk_array_get_active
tools.c, 305
tools.h, 309

HAVE_MPI
config.h, 67

id_experiment
Window, 47
id_experiment_name
Window, 48
id_input
Window, 48
id_template
Window, 48
id_variable
Window, 48
id_variable_label
Window, 48
INFO_TYPE
tools.h, 309
Input, 6
adaptation_ratio, 8
algorithm, 8
climbing, 8
directory, 8
evaluator, 9
experiment, 9
mutation_ratio, 9
name, 9
nbest, 9
nestimates, 10
nexperiments, 10
niterations, 10
norm, 10
nsimulations, 10
nsteps, 11
nvariables, 11
p, 11
relaxation, 11

reproduction_ratio, 11
result, 12
seed, 12
simulator, 12
threshold, 12
tolerance, 12
type, 13
variable, 13
variables, 13
input
input.c, 115
input.h, 132
input.c, 99
DEBUG_INPUT, 100
input, 115
input_error, 100
input_free, 101
input_new, 101
input_open, 102
input_open_json, 103
input_open_xml, 109
result_name, 115
variables_name, 115
input.h, 127
CLIMBING_METHOD_COORDINATES, 129
CLIMBING_METHOD_RANDOM, 129
ClimbingMethod, 128
ERROR_NORM_EUCLIDIAN, 129
ERROR_NORM_MAXIMUM, 129
ERROR_NORM_P, 129
ERROR_NORM_TAXICAB, 129
ErrorNorm, 129
input, 132
input_free, 129
input_new, 130
input_open, 130
result_name, 132
variables_name, 132
input_error
input.c, 100
INPUT_FILE
interface.c, 136
input_free
input.c, 101
input.h, 129
input_new
input.c, 101
input.h, 130
input_open
input.c, 102
input.h, 130
input_open_json
input.c, 103
input_open_xml
input.c, 109
input_save
interface.c, 143
input_save_climbing_json
interface.c, 145
input_save_climbing_xml
interface.c, 145
input_save_json
interface.c, 146
input_save_xml
interface.c, 149
INPUT_TYPE
config.h, 79
INPUT_TYPE_JSON
config.h, 79
INPUT_TYPE_XML
config.h, 79
interface.c, 134
DEBUG_INTERFACE, 136
dialog_evaluator, 137
dialog_evaluator_close, 137
dialog_name_experiment_close, 138
dialog_open_close, 138
dialog_options_close, 140
dialog_save_close, 140
dialog_simulator, 142
dialog_simulator_close, 143
INPUT_FILE, 136
input_save, 143
input_save_climbing_json, 145
input_save_climbing_xml, 145
input_save_json, 146
input_save_xml, 149
logo, 186
options, 186
options_new, 151
running, 187
running_new, 152
window, 187
window_about, 153
window_add_experiment, 153
window_add_variable, 154
window_get_algorithm, 155
window_get_climbing, 155
window_get_norm, 156
window_help, 157
window_inputs_experiment, 157
window_label_variable, 158
window_name_experiment, 158
window_new, 159
window_open, 169
window_precision_variable, 170
window_rangemax_variable, 171
window_rangemaxabs_variable, 171
window_rangemin_variable, 171
window_rangeminabs_variable, 171
window_read, 172
window_remove_experiment, 174
window_remove_variable, 175
window_run, 175
window_save, 176
window_save_climbing, 177

window_set_algorithm, 178
window_set_experiment, 179
window_set_variable, 179
window_step_variable, 180
window_template_experiment, 181
window_template_experiment_close, 182
window_update, 182
window_update_climbing, 185
window_update_variable, 185
window_weight_experiment, 186
interface.h, 220
MAX_LENGTH, 222
window, 232
window_new, 222

label
Optimize, 19
Running, 31
LABEL_ABSOLUTE_MAXIMUM
config.h, 67
LABEL_ABSOLUTE_MINIMUM
config.h, 67
LABEL_ADAPTATION
config.h, 67
label_adaptation
Window, 49
LABEL_ALGORITHM
config.h, 67
label_bests
Window, 49
label_bits
Window, 49
LABEL_CLIMBING
config.h, 68
label_climbing
Options, 28
LABEL_COORDINATES
config.h, 68
label_estimates
Window, 49
LABEL_EUCLIDIAN
config.h, 68
LABEL_EVALUATOR
config.h, 68
LABEL_EXPERIMENT
config.h, 68
label_experiment
Window, 49
LABEL_EXPERIMENTS
config.h, 69
label_generations
Window, 50
LABEL_GENETIC
config.h, 69
label_iterations
Window, 50
label_max
Window, 50
LABEL_MAXIMUM
config.h, 69
label_min
Window, 50
LABEL_MINIMUM
config.h, 69
LABEL_MONTE_CARLO
config.h, 69
LABEL_MUTATION
config.h, 70
label_mutation
Window, 50
LABEL_NAME
config.h, 70
LABEL_NBEST
config.h, 70
LABEL_NBITS
config.h, 70
LABEL_NESTIMATES
config.h, 70
LABEL_NGENERATIONS
config.h, 71
LABEL_NITERATIONS
config.h, 71
LABEL_NORM
config.h, 71
LABEL_NPOPULATION
config.h, 71
LABEL_NSIMULATIONS
config.h, 71
LABEL_NSTEPS
config.h, 72
LABEL_NSWEEPS
config.h, 72
LABEL_OPTIMIZE
config.h, 72
LABEL_ORTHOGONAL
config.h, 72
LABEL_P
config.h, 72
label_p
Window, 51
label_population
Window, 51
LABEL_PRECISION
config.h, 73
label_precision
Window, 51
LABEL_RANDOM
config.h, 73
LABEL_RELAXATION
config.h, 73
label_relaxation
Window, 51
LABEL_REPRODUCTION
config.h, 73
label_reproduction
Window, 51
label_result

Window, 52
LABEL_RESULT_FILE
config.h, 73
LABEL_SEED
config.h, 74
label_seed
Options, 28
label_simulations
Window, 52
LABEL_SIMULATOR
config.h, 74
label_simulator
Window, 52
LABEL_STEP
config.h, 74
label_step
Window, 52
label_steps
Window, 52
LABEL_SWEEP
config.h, 74
label_sweeps
Window, 53
LABEL_TAXICAB
config.h, 74
LABEL_TEMPLATE1
config.h, 75
LABEL_TEMPLATE2
config.h, 75
LABEL_TEMPLATE3
config.h, 75
LABEL_TEMPLATE4
config.h, 75
LABEL_TEMPLATE5
config.h, 75
LABEL_TEMPLATE6
config.h, 76
LABEL_TEMPLATE7
config.h, 76
LABEL_TEMPLATE8
config.h, 76
label_threads
Options, 28
LABEL_THRESHOLD
config.h, 76
label_threshold
Window, 53
LABEL_TOLERANCE
config.h, 76
label_tolerance
Window, 53
LABEL_VARIABLE
config.h, 77
label_variable
Window, 53
LABEL_VARIABLES
config.h, 77
label_variables
Window, 53
LABEL_VARIABLES_FILE
config.h, 77
LABEL_WEIGHT
config.h, 77
label_weight
Window, 54
LOCALE_DIR
config.h, 77
logo
interface.c, 186
Window, 54

main
main.c, 236
main.c, 235
main, 236
main_window
tools.c, 306
tools.h, 310
MAX_LENGTH
interface.h, 222
MAX_NINPUTS
config.h, 78
mpcotool
mpcotool.c, 239
mpcotool.h, 244
mpcotool.c, 237
DEBUG_MPCOTOOL, 238
mpcotool, 239
mpcotool.h, 243
mpcotool, 244
mpi_rank
Optimize, 19
mutation_ratio
Input, 9
Optimize, 19

NALGORITHMS
config.h, 78
name
Experiment, 5
Input, 9
Variable, 32
nbest
Input, 9
Optimize, 19
nbits
Optimize, 19
Variable, 32
NCLIMBINGS
config.h, 78
nend
Optimize, 20
nend_climbing
Optimize, 20
nestimates
Input, 10
Optimize, 20

nexperiments
Input, 10
Optimize, 20
Window, 54
ninputs
Experiment, 5
Optimize, 20
niterations
Input, 10
Optimize, 21
NNORMS
config.h, 78
norm
Input, 10
NPRECISIONS
config.h, 78
nsaveds
Optimize, 21
nsimulations
Input, 10
Optimize, 21
nstart
Optimize, 21
nstart_climbing
Optimize, 21
nsteps
Input, 11
Optimize, 22
nsweeps
Optimize, 22
Variable, 33
nthreads_climbing
optimize.c, 277
optimize.h, 303
nvariables
Input, 11
Optimize, 22
Window, 54

Optimize, 14
adaptation_ratio, 16
algorithm, 16
calculation_time, 17
climbing, 17
error_best, 17
error_old, 17
evaluator, 17
experiment, 18
file, 18
file_result, 18
file_variables, 18
genetic_variable, 18
label, 19
mpi_rank, 19
mutation_ratio, 19
nbest, 19
nbits, 19
nend, 20
nend_climbing, 20
nestimates, 20
nexperiments, 20
ninputs, 20
niterations, 21
nsaveds, 21
nsimulations, 21
nstart, 21
nstart_climbing, 21
nsteps, 22
nsweeps, 22
nvariables, 22
p, 22
precision, 22
rangemax, 23
rangemaxabs, 23
rangemin, 23
rangeminabs, 23
relaxation, 23
reproduction_ratio, 24
result, 24
rng, 24
seed, 24
simulation_best, 24
simulator, 25
step, 25
stop, 25
thread, 25
thread_climbing, 25
threshold, 26
tolerance, 26
value, 26
value_old, 26
variables, 26
weight, 27
optimize
optimize.c, 278
optimize.h, 303
optimize.c, 246
DEBUG_OPTIMIZE, 248
nthreads_climbing, 277
optimize, 278
optimize_algorithm, 278
optimize_best, 249
optimize_best_climbing, 250
optimize_climbing, 250
optimize_climbing_sequential, 251
optimize_climbing_thread, 252
optimize_estimate_climbing, 278
optimize_estimate_climbing_coordinates, 253
optimize_estimate_climbing_random, 254
optimize_free, 254
optimize_genetic, 254
optimize_genetic_objective, 255
optimize_input, 256
optimize_iterate, 257
optimize_merge, 258
optimize_merge_old, 259
optimize_MonteCarlo, 260

optimize_norm, 278
optimize_norm_euclidian, 260
optimize_norm_maximum, 261
optimize_norm_p, 262
optimize_norm_taxicab, 263
optimize_open, 263
optimize_orthogonal, 268
optimize_parse, 268
optimize_print, 270
optimize_refine, 270
optimize_save_old, 272
optimize_save_variables, 272
optimize_sequential, 273
optimize_step, 273
optimize_step_climbing, 274
optimize_sweep, 275
optimize_synchronise, 276
optimize_thread, 277
RM, 249
optimize.h, 297
nthreads_climbing, 303
optimize, 303
optimize_free, 298
optimize_open, 298
optimize_algorithm
optimize.c, 278
optimize_best
optimize.c, 249
optimize_best_climbing
optimize.c, 250
optimize_climbing
optimize.c, 250
optimize_climbing_sequential
optimize.c, 251
optimize_climbing_thread
optimize.c, 252
optimize_estimate_climbing
optimize.c, 278
optimize_estimate_climbing_coordinates
optimize.c, 253
optimize_estimate_climbing_random
optimize.c, 254
optimize_free
optimize.c, 254
optimize.h, 298
optimize_genetic
optimize.c, 254
optimize_genetic_objective
optimize.c, 255
optimize_input
optimize.c, 256
optimize_iterate
optimize.c, 257
optimize_merge
optimize.c, 258
optimize_merge_old
optimize.c, 259
optimize_MonteCarlo
optimize.c, 260
optimize_norm
optimize.c, 278
optimize_norm_euclidian
optimize.c, 260
optimize_norm_maximum
optimize.c, 261
optimize_norm_p
optimize.c, 262
optimize_norm_taxicab
optimize.c, 263
optimize_open
optimize.c, 263
optimize.h, 298
optimize_orthogonal
optimize.c, 268
optimize_parse
optimize.c, 268
optimize_print
optimize.c, 270
optimize_refine
optimize.c, 270
optimize_save_old
optimize.c, 272
optimize_save_variables
optimize.c, 272
optimize_sequential
optimize.c, 273
optimize_step
optimize.c, 273
optimize_step_climbing
optimize.c, 274
optimize_sweep
optimize.c, 275
optimize_synchronise
optimize.c, 276
optimize_thread
optimize.c, 277
Options, 27
dialog, 28
grid, 28
label_climbing, 28
label_seed, 28
label_threads, 28
spin_climbing, 29
spin_seed, 29
spin_threads, 29
options
interface.c, 186
options_new
interface.c, 151

p
Input, 11
Optimize, 22
ParallelData, 29
thread, 30
precision
Optimize, 22

Variable, 33
variable.c, 319
variable.h, 333
process_pending
tools.c, 306
tools.h, 310
PROGRAM_INTERFACE
config.h, 79

rangemax
Optimize, 23
Variable, 33
rangemaxabs
Optimize, 23
Variable, 33
rangemin
Optimize, 23
Variable, 33
rangeminabs
Optimize, 23
Variable, 34
relaxation
Input, 11
Optimize, 23
reproduction_ratio
Input, 11
Optimize, 24
result
Input, 12
Optimize, 24
result_name
input.c, 115
input.h, 132
RM
optimize.c, 249
rng
Optimize, 24
Running, 30
dialog, 31
grid, 31
label, 31
spinner, 31
running
interface.c, 187
running_new
interface.c, 152

scrolled_max
Window, 54
scrolled_maxabs
Window, 55
scrolled_min
Window, 55
scrolled_minabs
Window, 55
scrolled_p
Window, 55
scrolled_step
Window, 55
scrolled_threshold
Window, 56
seed
Input, 12
Optimize, 24
show_pending
tools.c, 306
tools.h, 310
simulation_best
Optimize, 24
simulator
Input, 12
Optimize, 25
spin_adaptation
Window, 56
spin_bests
Window, 56
spin_bits
Window, 56
spin_climbing
Options, 29
spin_estimates
Window, 56
spin_generations
Window, 57
spin_iterations
Window, 57
spin_max
Window, 57
spin_maxabs
Window, 57
spin_min
Window, 57
spin_minabs
Window, 58
spin_mutation
Window, 58
spin_p
Window, 58
spin_population
Window, 58
spin_precision
Window, 58
spin_relaxation
Window, 59
spin_reproduction
Window, 59
spin_seed
Options, 29
spin_simulations
Window, 59
spin_step
Window, 59
spin_steps
Window, 59
spin_sweeps
Window, 60
spin_threads

Options, 29
spin_threshold
Window, 60
spin_tolerance
Window, 60
spin_weight
Window, 60
spinner
Running, 31
stencil
Experiment, 6
experiment.c, 88
experiment.h, 98
step
Optimize, 25
Variable, 34
stop
Optimize, 25

thread
Optimize, 25
ParallelData, 30
thread_climbing
Optimize, 25
threshold
Input, 12
Optimize, 26
tolerance
Input, 12
Optimize, 26
tools.c, 304
error_message, 306
gtk_array_get_active, 305
main_window, 306
process_pending, 306
show_pending, 306
tools.h, 308
error_message, 310
ERROR_TYPE, 309
G_APPLICATION_DEFAULT_FLAGS, 309
gtk_array_get_active, 309
INFO_TYPE, 309
main_window, 310
process_pending, 310
show_pending, 310
type
Input, 13

value
Optimize, 26
value_old
Optimize, 26
Variable, 32
name, 32
nbits, 32
nsweeps, 33
precision, 33
rangemax, 33
rangemaxabs, 33
rangemin, 33
rangeminabs, 34
step, 34
variable
Input, 13
Window, 60
variable.c, 312
DEBUG_VARIABLE, 313
format, 319
precision, 319
variable_error, 313
variable_free, 313
variable_open_json, 314
variable_open_xml, 316
variable.h, 324
Algorithm, 326
ALGORITHM_GENETIC, 326
ALGORITHM_MONTE_CARLO, 326
ALGORITHM_ORTHOGONAL, 326
ALGORITHM_SWEEP, 326
format, 333
precision, 333
variable_error, 326
variable_free, 327
variable_open_json, 327
variable_open_xml, 330
variable_error
variable.c, 313
variable.h, 326
variable_free
variable.c, 313
variable.h, 327
variable_open_json
variable.c, 314
variable.h, 327
variable_open_xml
variable.c, 316
variable.h, 330
variables
Input, 13
Optimize, 26
variables_name
input.c, 115
input.h, 132

weight
Experiment, 6
Optimize, 27
Window, 34
application_directory, 39
box_buttons, 39
button_about, 39
button_add_experiment, 40
button_add_variable, 40
button_algorithm, 40
button_climbing, 40
button_evaluator, 40
button_exit, 41
button_experiment, 41

button_help, 41
button_norm, 41
button_open, 41
button_options, 42
button_remove_experiment, 42
button_remove_variable, 42
button_run, 42
button_save, 42
button_simulator, 43
button_template, 43
check_climbing, 43
check_evaluator, 43
check_maxabs, 43
check_minabs, 44
check_template, 44
combo_experiment, 44
combo_variable, 44
entry_result, 44
entry_variable, 45
entry_variables, 45
experiment, 45
frame_algorithm, 45
frame_experiment, 45
frame_norm, 46
frame_variable, 46
grid, 46
grid_algorithm, 46
grid_climbing, 46
grid_experiment, 47
grid_files, 47
grid_norm, 47
grid_variable, 47
id_experiment, 47
id_experiment_name, 48
id_input, 48
id_template, 48
id_variable, 48
id_variable_label, 48
label_adaptation, 49
label_bests, 49
label_bits, 49
label_estimates, 49
label_experiment, 49
label_generations, 50
label_iterations, 50
label_max, 50
label_min, 50
label_mutation, 50
label_p, 51
label_population, 51
label_precision, 51
label_relaxation, 51
label_reproduction, 51
label_result, 52
label_simulations, 52
label_simulator, 52
label_step, 52
label_steps, 52
label_sweeps, 53
label_threshold, 53
label_tolerance, 53
label_variable, 53
label_variables, 53
label_weight, 54
logo, 54
nexperiments, 54
nvariables, 54
scrolled_max, 54
scrolled_maxabs, 55
scrolled_min, 55
scrolled_minabs, 55
scrolled_p, 55
scrolled_step, 55
scrolled_threshold, 56
spin_adaptation, 56
spin_bests, 56
spin_bits, 56
spin_estimates, 56
spin_generations, 57
spin_iterations, 57
spin_max, 57
spin_maxabs, 57
spin_min, 57
spin_minabs, 58
spin_mutation, 58
spin_p, 58
spin_population, 58
spin_precision, 58
spin_relaxation, 59
spin_reproduction, 59
spin_simulations, 59
spin_step, 59
spin_steps, 59
spin_sweeps, 60
spin_threshold, 60
spin_tolerance, 60
spin_weight, 60
variable, 60
window, 61
window
interface.c, 187
interface.h, 232
Window, 61
window_about
interface.c, 153
window_add_experiment
interface.c, 153
window_add_variable
interface.c, 154
window_get_algorithm
interface.c, 155
window_get_climbing
interface.c, 155
window_get_norm
interface.c, 156
window_help

interface.c, 157
window_inputs_experiment
interface.c, 157
window_label_variable
interface.c, 158
window_name_experiment
interface.c, 158
window_new
interface.c, 159
interface.h, 222
window_open
interface.c, 169
window_precision_variable
interface.c, 170
window_rangemax_variable
interface.c, 171
window_rangemaxabs_variable
interface.c, 171
window_rangemin_variable
interface.c, 171
window_rangeminabs_variable
interface.c, 171
window_read
interface.c, 172
window_remove_experiment
interface.c, 174
window_remove_variable
interface.c, 175
window_run
interface.c, 175
window_save
interface.c, 176
window_save_climbing
interface.c, 177
window_set_algorithm
interface.c, 178
window_set_experiment
interface.c, 179
window_set_variable
interface.c, 179
window_step_variable
interface.c, 180
window_template_experiment
interface.c, 181
window_template_experiment_close
interface.c, 182
window_update
interface.c, 182
window_update_climbing
interface.c, 185
window_update_variable
interface.c, 185
window_weight_experiment
interface.c, 186