

MPCOTool

4.0.1

Generated by Doxygen 1.8.13

Contents

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
3.2	Input Struct Reference	6
3.2.1	Detailed Description	7
3.3	Optimize Struct Reference	7
3.3.1	Detailed Description	10
3.3.2	Field Documentation	10
3.3.2.1	thread_climbing	10
3.4	Options Struct Reference	11
3.4.1	Detailed Description	11
3.5	ParallelData Struct Reference	11
3.5.1	Detailed Description	12
3.6	Running Struct Reference	12
3.6.1	Detailed Description	12
3.7	Variable Struct Reference	12
3.7.1	Detailed Description	13
3.8	Window Struct Reference	13
3.8.1	Detailed Description	18

4	File Documentation	19
4.1	config.h File Reference	19
4.1.1	Detailed Description	22
4.1.2	Enumeration Type Documentation	22
4.1.2.1	INPUT_TYPE	22
4.2	config.h	23
4.3	experiment.c File Reference	24
4.3.1	Detailed Description	25
4.3.2	Function Documentation	25
4.3.2.1	experiment_error()	25
4.3.2.2	experiment_free()	26
4.3.2.3	experiment_new()	27
4.3.2.4	experiment_open_json()	27
4.3.2.5	experiment_open_xml()	29
4.3.3	Variable Documentation	31
4.3.3.1	stencil	31
4.4	experiment.c	31
4.5	experiment.h File Reference	35
4.5.1	Detailed Description	36
4.5.2	Function Documentation	36
4.5.2.1	experiment_error()	36
4.5.2.2	experiment_free()	36
4.5.2.3	experiment_new()	37
4.5.2.4	experiment_open_json()	38
4.5.2.5	experiment_open_xml()	39
4.6	experiment.h	41
4.7	input.c File Reference	42
4.7.1	Detailed Description	43
4.7.2	Function Documentation	43
4.7.2.1	input_error()	43

4.7.2.2	input_free()	44
4.7.2.3	input_new()	45
4.7.2.4	input_open()	45
4.7.2.5	input_open_json()	46
4.7.2.6	input_open_xml()	52
4.8	input.c	57
4.9	input.h File Reference	69
4.9.1	Detailed Description	70
4.9.2	Enumeration Type Documentation	70
4.9.2.1	ClimbingMethod	70
4.9.2.2	ErrorNorm	71
4.9.3	Function Documentation	71
4.9.3.1	input_error()	71
4.9.3.2	input_free()	72
4.9.3.3	input_new()	73
4.9.3.4	input_open()	73
4.9.3.5	input_open_json()	74
4.9.3.6	input_open_xml()	80
4.10	input.h	85
4.11	interface.c File Reference	86
4.11.1	Detailed Description	88
4.11.2	Function Documentation	88
4.11.2.1	input_save()	88
4.11.2.2	input_save_climbing_json()	90
4.11.2.3	input_save_climbing_xml()	91
4.11.2.4	input_save_json()	92
4.11.2.5	input_save_xml()	95
4.11.2.6	options_new()	98
4.11.2.7	running_new()	99
4.11.2.8	window_about()	99

4.11.2.9	<code>window_add_experiment()</code>	100
4.11.2.10	<code>window_add_variable()</code>	101
4.11.2.11	<code>window_get_algorithm()</code>	102
4.11.2.12	<code>window_get_climbing()</code>	103
4.11.2.13	<code>window_get_norm()</code>	103
4.11.2.14	<code>window_help()</code>	104
4.11.2.15	<code>window_inputs_experiment()</code>	105
4.11.2.16	<code>window_label_variable()</code>	105
4.11.2.17	<code>window_name_experiment()</code>	106
4.11.2.18	<code>window_new()</code>	106
4.11.2.19	<code>window_open()</code>	115
4.11.2.20	<code>window_precision_variable()</code>	116
4.11.2.21	<code>window_rangemax_variable()</code>	117
4.11.2.22	<code>window_rangemaxabs_variable()</code>	117
4.11.2.23	<code>window_rangemin_variable()</code>	117
4.11.2.24	<code>window_rangeminabs_variable()</code>	118
4.11.2.25	<code>window_read()</code>	118
4.11.2.26	<code>window_remove_experiment()</code>	120
4.11.2.27	<code>window_remove_variable()</code>	121
4.11.2.28	<code>window_run()</code>	122
4.11.2.29	<code>window_save()</code>	123
4.11.2.30	<code>window_save_climbing()</code>	125
4.11.2.31	<code>window_set_algorithm()</code>	126
4.11.2.32	<code>window_set_experiment()</code>	127
4.11.2.33	<code>window_set_variable()</code>	127
4.11.2.34	<code>window_step_variable()</code>	129
4.11.2.35	<code>window_template_experiment()</code>	129
4.11.2.36	<code>window_update()</code>	130
4.11.2.37	<code>window_update_climbing()</code>	132
4.11.2.38	<code>window_update_variable()</code>	133

4.11.2.39 window_weight_experiment()	133
4.12 interface.c	134
4.13 interface.h File Reference	165
4.13.1 Detailed Description	167
4.13.2 Function Documentation	167
4.13.2.1 gtk_array_get_active()	167
4.13.2.2 input_save()	168
4.13.2.3 options_new()	169
4.13.2.4 running_new()	170
4.13.2.5 window_add_experiment()	171
4.13.2.6 window_add_variable()	172
4.13.2.7 window_get_algorithm()	173
4.13.2.8 window_get_climbing()	173
4.13.2.9 window_get_norm()	174
4.13.2.10 window_help()	175
4.13.2.11 window_inputs_experiment()	175
4.13.2.12 window_label_variable()	176
4.13.2.13 window_name_experiment()	176
4.13.2.14 window_new()	176
4.13.2.15 window_open()	185
4.13.2.16 window_precision_variable()	186
4.13.2.17 window_rangemax_variable()	187
4.13.2.18 window_rangemaxabs_variable()	187
4.13.2.19 window_rangemin_variable()	188
4.13.2.20 window_rangeminabs_variable()	188
4.13.2.21 window_read()	188
4.13.2.22 window_remove_experiment()	190
4.13.2.23 window_remove_variable()	191
4.13.2.24 window_run()	192
4.13.2.25 window_save()	193

4.13.2.26 window_save_climbing()	195
4.13.2.27 window_set_algorithm()	196
4.13.2.28 window_set_experiment()	197
4.13.2.29 window_set_variable()	198
4.13.2.30 window_template_experiment()	199
4.13.2.31 window_update()	200
4.13.2.32 window_update_climbing()	202
4.13.2.33 window_update_variable()	203
4.13.2.34 window_weight_experiment()	203
4.14 interface.h	204
4.15 main.c File Reference	206
4.15.1 Detailed Description	207
4.16 main.c	207
4.17 mpcotool.c File Reference	208
4.17.1 Detailed Description	209
4.17.2 Function Documentation	209
4.17.2.1 mpcotool()	209
4.18 mpcotool.c	212
4.19 optimize.c File Reference	214
4.19.1 Detailed Description	216
4.19.2 Function Documentation	216
4.19.2.1 optimize_best()	216
4.19.2.2 optimize_best_climbing()	217
4.19.2.3 optimize_climbing()	218
4.19.2.4 optimize_climbing_sequential()	219
4.19.2.5 optimize_climbing_thread()	220
4.19.2.6 optimize_estimate_climbing_coordinates()	221
4.19.2.7 optimize_estimate_climbing_random()	221
4.19.2.8 optimize_free()	222
4.19.2.9 optimize_genetic()	223

4.19.2.10 optimize_genetic_objective()	223
4.19.2.11 optimize_input()	224
4.19.2.12 optimize_iterate()	226
4.19.2.13 optimize_merge()	226
4.19.2.14 optimize_merge_old()	227
4.19.2.15 optimize_MonteCarlo()	228
4.19.2.16 optimize_norm_euclidian()	229
4.19.2.17 optimize_norm_maximum()	230
4.19.2.18 optimize_norm_p()	230
4.19.2.19 optimize_norm_taxicab()	231
4.19.2.20 optimize_open()	232
4.19.2.21 optimize_orthogonal()	237
4.19.2.22 optimize_parse()	237
4.19.2.23 optimize_print()	239
4.19.2.24 optimize_refine()	240
4.19.2.25 optimize_save_old()	241
4.19.2.26 optimize_save_variables()	241
4.19.2.27 optimize_sequential()	242
4.19.2.28 optimize_step()	243
4.19.2.29 optimize_step_climbing()	243
4.19.2.30 optimize_sweep()	244
4.19.2.31 optimize_synchronise()	245
4.19.2.32 optimize_thread()	246
4.20 optimize.c	247
4.21 optimize.h File Reference	265
4.21.1 Detailed Description	267
4.21.2 Function Documentation	267
4.21.2.1 optimize_best()	267
4.21.2.2 optimize_best_climbing()	268
4.21.2.3 optimize_climbing()	268

4.21.2.4	<code>optimize_climbing_sequential()</code>	270
4.21.2.5	<code>optimize_climbing_thread()</code>	271
4.21.2.6	<code>optimize_estimate_climbing_coordinates()</code>	271
4.21.2.7	<code>optimize_estimate_climbing_random()</code>	272
4.21.2.8	<code>optimize_free()</code>	273
4.21.2.9	<code>optimize_genetic()</code>	273
4.21.2.10	<code>optimize_genetic_objective()</code>	274
4.21.2.11	<code>optimize_input()</code>	275
4.21.2.12	<code>optimize_iterate()</code>	276
4.21.2.13	<code>optimize_merge()</code>	277
4.21.2.14	<code>optimize_merge_old()</code>	278
4.21.2.15	<code>optimize_MonteCarlo()</code>	279
4.21.2.16	<code>optimize_norm_euclidian()</code>	279
4.21.2.17	<code>optimize_norm_maximum()</code>	280
4.21.2.18	<code>optimize_norm_p()</code>	281
4.21.2.19	<code>optimize_norm_taxicab()</code>	282
4.21.2.20	<code>optimize_open()</code>	283
4.21.2.21	<code>optimize_orthogonal()</code>	287
4.21.2.22	<code>optimize_parse()</code>	288
4.21.2.23	<code>optimize_print()</code>	290
4.21.2.24	<code>optimize_refine()</code>	291
4.21.2.25	<code>optimize_save_old()</code>	292
4.21.2.26	<code>optimize_save_variables()</code>	292
4.21.2.27	<code>optimize_sequential()</code>	293
4.21.2.28	<code>optimize_step()</code>	294
4.21.2.29	<code>optimize_step_climbing()</code>	294
4.21.2.30	<code>optimize_sweep()</code>	295
4.21.2.31	<code>optimize_synchronise()</code>	296
4.21.2.32	<code>optimize_thread()</code>	297
4.22	<code>optimize.h</code>	298

4.23	utils.c File Reference	300
4.23.1	Detailed Description	301
4.23.2	Function Documentation	301
4.23.2.1	cores_number()	301
4.23.2.2	gtk_array_get_active()	301
4.23.2.3	json_object_get_float()	302
4.23.2.4	json_object_get_float_with_default()	303
4.23.2.5	json_object_get_int()	303
4.23.2.6	json_object_get_uint()	304
4.23.2.7	json_object_get_uint_with_default()	305
4.23.2.8	json_object_set_float()	306
4.23.2.9	json_object_set_int()	306
4.23.2.10	json_object_set_uint()	307
4.23.2.11	process_pending()	307
4.23.2.12	show_error()	307
4.23.2.13	show_message()	308
4.23.2.14	xml_node_get_float()	309
4.23.2.15	xml_node_get_float_with_default()	309
4.23.2.16	xml_node_get_int()	310
4.23.2.17	xml_node_get_uint()	311
4.23.2.18	xml_node_get_uint_with_default()	312
4.23.2.19	xml_node_set_float()	312
4.23.2.20	xml_node_set_int()	313
4.23.2.21	xml_node_set_uint()	313
4.24	utils.c	314
4.25	utils.h File Reference	318
4.25.1	Detailed Description	319
4.25.2	Function Documentation	320
4.25.2.1	cores_number()	320
4.25.2.2	gtk_array_get_active()	320

4.25.2.3	json_object_get_float()	321
4.25.2.4	json_object_get_float_with_default()	321
4.25.2.5	json_object_get_int()	322
4.25.2.6	json_object_get_uint()	323
4.25.2.7	json_object_get_uint_with_default()	324
4.25.2.8	json_object_set_float()	324
4.25.2.9	json_object_set_int()	325
4.25.2.10	json_object_set_uint()	325
4.25.2.11	process_pending()	326
4.25.2.12	show_error()	326
4.25.2.13	show_message()	327
4.25.2.14	xml_node_get_float()	328
4.25.2.15	xml_node_get_float_with_default()	328
4.25.2.16	xml_node_get_int()	329
4.25.2.17	xml_node_get_uint()	330
4.25.2.18	xml_node_get_uint_with_default()	331
4.25.2.19	xml_node_set_float()	331
4.25.2.20	xml_node_set_int()	332
4.25.2.21	xml_node_set_uint()	332
4.26	utils.h	333
4.27	variable.c File Reference	334
4.27.1	Detailed Description	335
4.27.2	Function Documentation	335
4.27.2.1	variable_error()	335
4.27.2.2	variable_free()	336
4.27.2.3	variable_new()	336
4.27.2.4	variable_open_json()	337
4.27.2.5	variable_open_xml()	339
4.27.3	Variable Documentation	342
4.27.3.1	format	342
4.27.3.2	precision	342
4.28	variable.c	342
4.29	variable.h File Reference	347
4.29.1	Detailed Description	348
4.29.2	Enumeration Type Documentation	348
4.29.2.1	Algorithm	348
4.29.3	Function Documentation	349
4.29.3.1	variable_error()	349
4.29.3.2	variable_free()	349
4.29.3.3	variable_new()	350
4.29.3.4	variable_open_json()	350
4.29.3.5	variable_open_xml()	353
4.30	variable.h	355

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	7
Options	Struct to define the options dialog	11
ParallelData	Struct to pass to the GThreads parallelized function	11
Running	Struct to define the running dialog	12
Variable	Struct to define the variable data	12
Window	Struct to define the main window	13

Chapter 2

File Index

2.1 File List

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

config.h	Configuration header file	19
experiment.c	Source file to define the experiment data	24
experiment.h	Header file to define the experiment data	35
input.c	Source file to define the input functions	42
input.h	Header file to define the input functions	69
interface.c	Source file to define the graphical interface functions	86
interface.h	Header file to define the graphical interface functions	165
main.c	Main source file	206
mpcotool.c	Main function source file	208
mpcotool.h	??
optimize.c	Source file to define the optimization functions	214
optimize.h	Header file to define the optimization functions	265
utils.c	Source file to define some useful functions	300
utils.h	Header file to define some useful functions	318
variable.c	Source file to define the variable data	334
variable.h	Header file to define the variable data	347

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](#).

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

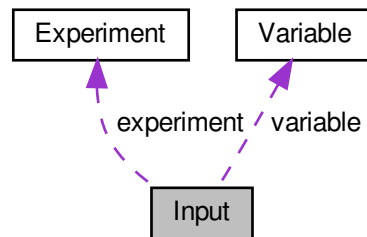
- [experiment.h](#)

3.2 Input Struct Reference

Struct to define the optimization input file.

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

Collaboration diagram for Input:



Data Fields

- [Experiment](#) * [experiment](#)
Array of 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](#).

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

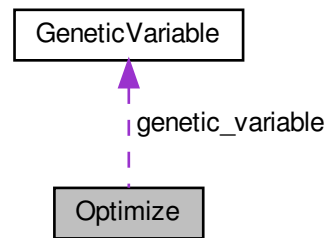
- [input.h](#)

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 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](#).

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](#).

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](#).

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](#).

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](#).

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

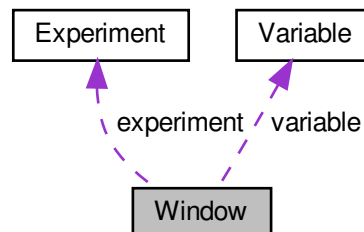
- [variable.h](#)

3.8 Window Struct Reference

Struct to define the main window.

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

Collaboration diagram for Window:



Data Fields

- GtkWidget * [window](#)
Main GtkWidget.
- GtkWidget * [grid](#)
Main GtkWidget.
- GtkWidget * [bar_buttons](#)
GtkToolbar to store the main buttons.
- GtkWidget * [button_open](#)
Open GtkWidget.
- GtkWidget * [button_save](#)
Save GtkWidget.
- GtkWidget * [button_run](#)
Run GtkWidget.
- GtkWidget * [button_options](#)
Options GtkWidget.
- GtkWidget * [button_help](#)
Help GtkWidget.
- GtkWidget * [button_about](#)
Help GtkWidget.
- GtkWidget * [button_exit](#)
Exit GtkWidget.
- GtkWidget * [grid_files](#)
Files GtkWidget.
- GtkWidget * [label_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [button_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [check_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [button_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [label_result](#)
Result file GtkWidget.
- GtkWidget * [entry_result](#)
Result file GtkWidget.
- GtkWidget * [label_variables](#)
Variables file GtkWidget.
- GtkWidget * [entry_variables](#)
Variables file GtkWidget.
- GtkWidget * [frame_norm](#)
GtkFrame to set the error norm.
- GtkWidget * [grid_norm](#)
GtkGrid to set the error norm.
- GtkWidget * [button_norm](#) [NNORMS]
Array of GtkButtons to set the error norm.
- GtkWidget * [label_p](#)
GtkLabel to set the p parameter.
- GtkWidget * [spin_p](#)
GtkSpinButton to set the p parameter.
- GtkWidget * [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 *GtkButtons* 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_best`
GtkLabel to set the best number.
- `GtkSpinButton * spin_best`
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]
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.*
- GtkWidget * [scrolled_maxabs](#)
 - Absolute maximum GtkWidget.*
- GtkWidget * [label_precision](#)
 - Precision GtkWidget.*
- GtkWidget * [spin_precision](#)
 - Precision digits GtkWidget.*
- GtkWidget * [label_sweeps](#)
 - Sweeps number GtkWidget.*
- GtkWidget * [spin_sweeps](#)
 - Sweeps number GtkWidget.*
- GtkWidget * [label_bits](#)
 - Bits number GtkWidget.*
- GtkWidget * [spin_bits](#)
 - Bits number GtkWidget.*
- GtkWidget * [label_step](#)
 - GtkWidget to set the step.*
- GtkWidget * [spin_step](#)
 - GtkWidget to set the step.*
- GtkWidget * [scrolled_step](#)
 - step GtkWidget.*
- GtkWidget * [frame_experiment](#)
 - Experiment GtkWidget.*
- GtkWidget * [grid_experiment](#)
 - Experiment GtkWidget.*
- GtkWidget * [combo_experiment](#)
 - Experiment GtkWidgetEntry.*
- GtkWidget * [button_add_experiment](#)
 - GtkWidget to add a experiment.*
- GtkWidget * [button_remove_experiment](#)
 - GtkWidget to remove a experiment.*
- GtkWidget * [label_experiment](#)
 - Experiment GtkWidget.*
- GtkWidget * [button_experiment](#)
 - GtkFileChooserButton to set the experimental data file.*
- GtkWidget * [label_weight](#)
 - Weight GtkWidget.*
- GtkWidget * [spin_weight](#)
 - Weight GtkWidget.*
- GtkWidget * [check_template](#) [MAX_NINPUTS]
 - Array of GtkWidgetButtons to set the input templates.*
- GtkWidget * [button_template](#) [MAX_NINPUTS]
 - Array of GtkFileChooserButtons 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](#).

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

- [interface.h](#)

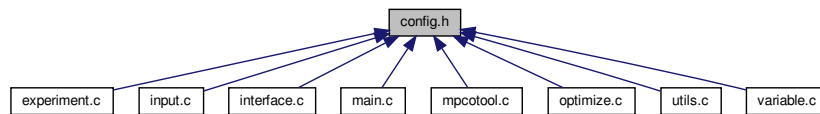
Chapter 4

File Documentation

4.1 config.h File Reference

Configuration header file.

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



Macros

- `#define _(string) (gettext(string))`
- `#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 Enumeration Type Documentation

4.1.2.1 INPUT_TYPE

`enum INPUT_TYPE`

Enum to define the input file types.

Enumerator

<code>INPUT_TYPE_XML</code>	XML input file.
<code>INPUT_TYPE_JSON</code>	JSON input file.

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

```

00127 {
00128     INPUT_TYPE_XML = 0,
00129     INPUT_TYPE_JSON = 1
00130 };

```

4.2 config.h

```

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 // Gettext simplification
00043 #define _(string) (gettext(string))
00044
00045 // Array sizes
00046
00047 #define MAX_NINPUTS 8
00048 #define NALGORITHMS 4
00050 #define NCLIMBINGS 2
00051 #define NNORMS 4
00052 #define NPRECISIONS 15
00053
00054 // Default choices
00055
00056 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00057 #define DEFAULT_RANDOM_SEED 7007
00058 #define DEFAULT_RELAXATION 1.
00059
00060 // Interface labels
00061
00062 #define LOCALE_DIR "locales"
00063 #define PROGRAM_INTERFACE "mpcotool"
00064
00065 // Labels
00066
00067 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00068 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00070 #define LABEL_ADAPTATION "adaptation"
00072 #define LABEL_ALGORITHM "algorithm"
00073 #define LABEL_CLIMBING "climbing"
00074 #define LABEL_COORDINATES "coordinates"
00075 #define LABEL_EUCLIDIAN "euclidian"
00076 #define LABEL_EVALUATOR "evaluator"
00077 #define LABEL_EXPERIMENT "experiment"
00078 #define LABEL_EXPERIMENTS "experiments"
00079 #define LABEL_GENETIC "genetic"
00080 #define LABEL_MINIMUM "minimum"
00081 #define LABEL_MAXIMUM "maximum"
00082 #define LABEL_MONTE_CARLO "Monte-Carlo"
00083 #define LABEL_MUTATION "mutation"
00084 #define LABEL_NAME "name"

```

```

00085 #define LABEL_NBEST "nbest"
00086 #define LABEL_NBITS "nbits"
00087 #define LABEL_NESTIMATES "nestimates"
00088 #define LABEL_NGENERATIONS "ngenerations"
00089 #define LABEL_NITERATIONS "niterations"
00090 #define LABEL_NORM "norm"
00091 #define LABEL_NPOPULATION "npopulation"
00092 #define LABEL_NSIMULATIONS "nsimulations"
00093 #define LABEL_NSTEPS "nsteps"
00094 #define LABEL_NSWEEPS "nsweeps"
00095 #define LABEL_OPTIMIZE "optimize"
00096 #define LABEL_ORTHOGONAL "orthogonal"
00097 #define LABEL_P "p"
00098 #define LABEL_PRECISION "precision"
00099 #define LABEL_RANDOM "random"
00100 #define LABEL_RELAXATION "relaxation"
00101 #define LABEL_REPRODUCTION "reproduction"
00102 #define LABEL_RESULT_FILE "result_file"
00103 #define LABEL_SIMULATOR "simulator"
00104 #define LABEL_SEED "seed"
00105 #define LABEL_STEP "step"
00106 #define LABEL_SWEEP "sweep"
00107 #define LABEL_TAXICAB "taxicab"
00108 #define LABEL_TEMPLATE1 "template1"
00109 #define LABEL_TEMPLATE2 "template2"
00110 #define LABEL_TEMPLATE3 "template3"
00111 #define LABEL_TEMPLATE4 "template4"
00112 #define LABEL_TEMPLATE5 "template5"
00113 #define LABEL_TEMPLATE6 "template6"
00114 #define LABEL_TEMPLATE7 "template7"
00115 #define LABEL_TEMPLATE8 "template8"
00116 #define LABEL_THRESHOLD "threshold"
00117 #define LABEL_TOLERANCE "tolerance"
00118 #define LABEL_VARIABLE "variable"
00119 #define LABEL_VARIABLES "variables"
00120 #define LABEL_VARIABLES_FILE "variables_file"
00121 #define LABEL_WEIGHT "weight"
00122
00123 // Enumerations
00124
00126 enum INPUT_TYPE
00127 {
00128     INPUT_TYPE_XML = 0,
00129     INPUT_TYPE_JSON = 1
00130 };
00131
00132 #endif

```

4.3 experiment.c File Reference

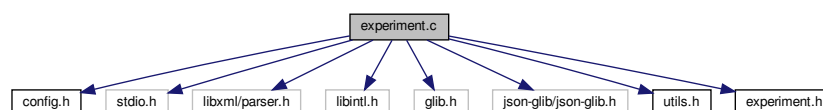
Source file to define the experiment data.

```

#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "experiment.h"

```

Include dependency graph for experiment.c:



Macros

- `#define DEBUG_EXPERIMENT 0`
Macro to debug experiment functions.

Functions

- 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

Copyright 2012-2018, all rights reserved.

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

4.3.2 Function Documentation

4.3.2.1 [experiment_error\(\)](#)

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>message</i>	Error message.

Definition at line 109 of file [experiment.c](#).

```

00111 {
00112     char buffer[64];
00113     if (!experiment->name)
00114         snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00117                 experiment->name, message);
00118     error_message = g_strdup (buffer);
00119 }
```

4.3.2.2 `experiment_free()`

```

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>type</i>	Type of input file.

Definition at line 80 of file [experiment.c](#).

```

00082 {
00083     unsigned int i;
00084     #if DEBUG_EXPERIMENT
00085     fprintf (stderr, "experiment_free: start\n");
00086     #endif
00087     if (type == INPUT_TYPE_XML)
00088     {
00089         for (i = 0; i < experiment->ninputs; ++i)
00090             xmlFree (experiment->stencil[i]);
00091         xmlFree (experiment->name);
00092     }
00093     else
00094     {
00095         for (i = 0; i < experiment->ninputs; ++i)
00096             g_free (experiment->stencil[i]);
00097         g_free (experiment->name);
00098     }
00099     experiment->ninputs = 0;
00100     #if DEBUG_EXPERIMENT
00101     fprintf (stderr, "experiment_free: end\n");
00102     #endif
00103 }
```


4.3.2.3 experiment_new()

```
void experiment_new (
    Experiment * experiment )
```

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

Parameters

<i>experiment</i>	Experiment struct.
-------------------	------------------------------------

Definition at line 61 of file [experiment.c](#).

```
00062 {
00063     unsigned int i;
00064     #if DEBUG_EXPERIMENT
00065     fprintf (stderr, "experiment_new: start\n");
00066     #endif
00067     experiment->name = NULL;
00068     experiment->ninputs = 0;
00069     for (i = 0; i < MAX_NINPUTS; ++i)
00070         experiment->stencil[i] = NULL;
00071     #if DEBUG_EXPERIMENT
00072     fprintf (stderr, "input_new: end\n");
00073     #endif
00074 }
```

4.3.2.4 experiment_open_json()

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

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

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 231 of file [experiment.c](#).

```
00235 {
00236     char buffer[64];
00237     JsonObject *object;
00238     const char *name;
00239     int error_code;
```

```

00240     unsigned int i;
00241
00242     #if DEBUG_EXPERIMENT
00243         fprintf (stderr, "experiment_open_json: start\n");
00244     #endif
00245
00246     // Resetting experiment data
00247     experiment_new (experiment);
00248
00249     // Getting JSON object
00250     object = json_node_get_object (node);
00251
00252     // Reading the experimental data
00253     name = json_object_get_string_member (object, LABEL_NAME);
00254     if (!name)
00255     {
00256         experiment_error (experiment, _("no data file name"));
00257         goto exit_on_error;
00258     }
00259     experiment->name = g_strdup (name);
00260     #if DEBUG_EXPERIMENT
00261         fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00262     #endif
00263     experiment->weight
00264     = json_object_get_float_with_default (object,
00265     LABEL_WEIGHT, 1.,
00266     &error_code);
00267     if (error_code)
00268     {
00269         experiment_error (experiment, _("bad weight"));
00270         goto exit_on_error;
00271     }
00272     #if DEBUG_EXPERIMENT
00273         fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00274     #endif
00275     name = json_object_get_string_member (object, stencil[0]);
00276     if (name)
00277     {
00278         #if DEBUG_EXPERIMENT
00279             fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00280             name, stencil[0]);
00281         #endif
00282         ++experiment->ninputs;
00283     }
00284     else
00285     {
00286         experiment_error (experiment, _("no template"));
00287         goto exit_on_error;
00288     }
00289     experiment->stencil[0] = g_strdup (name);
00290     for (i = 1; i < MAX_NINPUTS; ++i)
00291     {
00292         #if DEBUG_EXPERIMENT
00293             fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00294         #endif
00295         if (json_object_get_member (object, stencil[i]))
00296         {
00297             if (ninputs && ninputs <= i)
00298             {
00299                 experiment_error (experiment, _("bad templates number"));
00300                 goto exit_on_error;
00301             }
00302             name = json_object_get_string_member (object, stencil[i]);
00303             #if DEBUG_EXPERIMENT
00304                 fprintf (stderr,
00305                 "experiment_open_json: experiment=%s stencil%u=%s\n",
00306                 experiment->nexperiments, name, stencil[i]);
00307             #endif
00308             experiment->stencil[i] = g_strdup (name);
00309             ++experiment->ninputs;
00310         }
00311         else if (ninputs && ninputs > i)
00312         {
00313             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00314             experiment_error (experiment, buffer);
00315             goto exit_on_error;
00316         }
00317         else
00318             break;
00319     }
00320     #if DEBUG_EXPERIMENT
00321         fprintf (stderr, "experiment_open_json: end\n");
00322     #endif
00323     return 1;
00324
00325 exit_on_error:

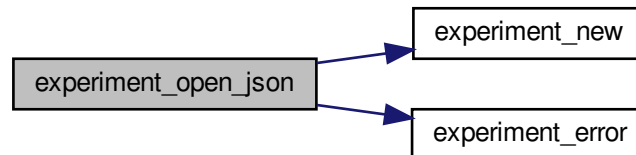
```

```

00326     experiment_free (experiment, INPUT_TYPE_JSON);
00327     #if DEBUG_EXPERIMENT
00328     fprintf (stderr, "experiment_open_json: end\n");
00329     #endif
00330     return 0;
00331 }

```

Here is the call graph for this function:



4.3.2.5 experiment_open_xml()

```

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

```

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

Returns

1 on success, 0 on error.

Parameters

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

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

```

00131 {
00132     char buffer[64];
00133     int error_code;
00134     unsigned int i;
00135
00136     #if DEBUG_EXPERIMENT
00137     fprintf (stderr, "experiment_open_xml: start\n");
00138     #endif
00139
00140     // Resetting experiment data
00141     experiment_new (experiment);

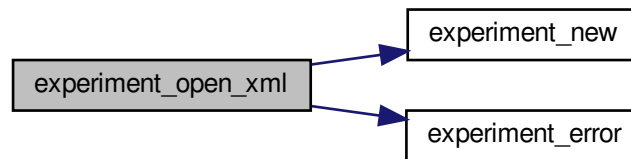
```

```

00142
00143 // Reading the experimental data
00144 experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145 if (!experiment->name)
00146 {
00147     experiment_error (experiment, _("no data file name"));
00148     goto exit_on_error;
00149 }
00150 #if DEBUG_EXPERIMENT
00151 fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00152 #endif
00153 experiment->weight
00154 =
00155     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
00156                                     &error_code);
00157 if (error_code)
00158 {
00159     experiment_error (experiment, _("bad weight"));
00160     goto exit_on_error;
00161 }
00162 #if DEBUG_EXPERIMENT
00163 fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00164 #endif
00165 experiment->stencil[0]
00166 = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00167 if (experiment->stencil[0])
00168 {
00169     #if DEBUG_EXPERIMENT
00170     fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
experiment->name, stencil[0]);
00171     #endif
00172     ++experiment->ninputs;
00173 }
00174 else
00175 {
00176     experiment_error (experiment, _("no template"));
00177     goto exit_on_error;
00178 }
00179 for (i = 1; i < MAX_NINPUTS; ++i)
00180 {
00181     #if DEBUG_EXPERIMENT
00182     fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00183     #endif
00184     if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00185     {
00186         if (ninputs && ninputs <= i)
00187         {
00188             experiment_error (experiment, _("bad templates number"));
00189             goto exit_on_error;
00190         }
00191         experiment->stencil[i]
00192         = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00193     }
00194     #if DEBUG_EXPERIMENT
00195     fprintf (stderr,
"experiment_open_xml: experiment=%s stencil%u=%s\n",
experiment->nexperiments, experiment->name,
experiment->stencil[i]);
00196     #endif
00197     ++experiment->ninputs;
00198 }
00199 else if (ninputs && ninputs > i)
00200 {
00201     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00202     experiment_error (experiment, buffer);
00203     goto exit_on_error;
00204 }
00205 else
00206     break;
00207 }
00208 #if DEBUG_EXPERIMENT
00209 fprintf (stderr, "experiment_open_xml: end\n");
00210 #endif
00211 return 1;
00212 exit_on_error:
00213 experiment_free (experiment, INPUT_TYPE_XML);
00214 #if DEBUG_EXPERIMENT
00215 fprintf (stderr, "experiment_open_xml: end\n");
00216 #endif
00217 return 0;
00218 }

```

Here is the call graph for this function:



4.3.3 Variable Documentation

4.3.3.1 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 50 of file [experiment.c](#).

4.4 experiment.c

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

```

```

00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "experiment.h"
00041
00042 #define DEBUG_EXPERIMENT 0
00043
00044 const char *stencil[MAX_NINPUTS] = {
00045     LABEL_TEMPLATE1, LABEL_TEMPLATE2,
00046     LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00047     LABEL_TEMPLATE5, LABEL_TEMPLATE6,
00048     LABEL_TEMPLATE7, LABEL_TEMPLATE8
00049 };
00050
00051 void
00052 experiment_new (Experiment * experiment)
00053 {
00054     unsigned int i;
00055     #if DEBUG_EXPERIMENT
00056         fprintf (stderr, "experiment_new: start\n");
00057     #endif
00058     experiment->name = NULL;
00059     experiment->ninputs = 0;
00060     for (i = 0; i < MAX_NINPUTS; ++i)
00061         experiment->stencil[i] = NULL;
00062     #if DEBUG_EXPERIMENT
00063         fprintf (stderr, "input_new: end\n");
00064     #endif
00065 }
00066
00067 void
00068 experiment_free (Experiment * experiment,
00069                 unsigned int type)
00070 {
00071     unsigned int i;
00072     #if DEBUG_EXPERIMENT
00073         fprintf (stderr, "experiment_free: start\n");
00074     #endif
00075     if (type == INPUT_TYPE_XML)
00076     {
00077         for (i = 0; i < experiment->ninputs; ++i)
00078             xmlFree (experiment->stencil[i]);
00079         xmlFree (experiment->name);
00080     }
00081     else
00082     {
00083         for (i = 0; i < experiment->ninputs; ++i)
00084             g_free (experiment->stencil[i]);
00085         g_free (experiment->name);
00086     }
00087     experiment->ninputs = 0;
00088     #if DEBUG_EXPERIMENT
00089         fprintf (stderr, "experiment_free: end\n");
00090     #endif
00091 }
00092
00093 void
00094 experiment_error (Experiment * experiment,
00095                  char *message)
00096 {
00097     char buffer[64];
00098     if (!experiment->name)
00099         snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00100     else
00101         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00102                  experiment->name, message);
00103     error_message = g_strdup (buffer);
00104 }
00105
00106 int
00107 experiment_open_xml (Experiment * experiment,

```

```

00128             xmlNode * node,
00129             unsigned int ninputs)
00131 {
00132     char buffer[64];
00133     int error_code;
00134     unsigned int i;
00135
00136     #if DEBUG_EXPERIMENT
00137         fprintf (stderr, "experiment_open_xml: start\n");
00138     #endif
00139
00140     // Resetting experiment data
00141     experiment_new (experiment);
00142
00143     // Reading the experimental data
00144     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145     if (!experiment->name)
00146     {
00147         experiment_error (experiment, _("no data file name"));
00148         goto exit_on_error;
00149     }
00150     #if DEBUG_EXPERIMENT
00151         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00152     #endif
00153     experiment->weight
00154         =
00155         xml_node_get_float_with_default (node, (const xmlChar *)
00156         LABEL_WEIGHT, 1.,
00157                                         &error_code);
00158     if (error_code)
00159     {
00160         experiment_error (experiment, _("bad weight"));
00161         goto exit_on_error;
00162     }
00163     #if DEBUG_EXPERIMENT
00164         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00165     #endif
00166     experiment->stencil[0]
00167         = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00168     if (experiment->stencil[0])
00169     {
00170         #if DEBUG_EXPERIMENT
00171             fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00172                     experiment->name, stencil[0]);
00173         #endif
00174         ++experiment->ninputs;
00175     }
00176     else
00177     {
00178         experiment_error (experiment, _("no template"));
00179         goto exit_on_error;
00180     }
00181     for (i = 1; i < MAX_NINPUTS; ++i)
00182     {
00183         #if DEBUG_EXPERIMENT
00184             fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00185         #endif
00186         if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00187         {
00188             if (ninputs && ninputs <= i)
00189             {
00190                 experiment_error (experiment, _("bad templates number"));
00191                 goto exit_on_error;
00192             }
00193             experiment->stencil[i]
00194                 = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00195             #if DEBUG_EXPERIMENT
00196                 fprintf (stderr,
00197                         "experiment_open_xml: experiment=%s stencil%u=%s\n",
00198                         experiment->nexperiments, experiment->name,
00199                         experiment->stencil[i]);
00200             #endif
00201             ++experiment->ninputs;
00202         }
00203         else if (ninputs && ninputs > i)
00204         {
00205             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00206             experiment_error (experiment, buffer);
00207             goto exit_on_error;
00208         }
00209         else
00210             break;
00211     }
00212     #if DEBUG_EXPERIMENT
00213         fprintf (stderr, "experiment_open_xml: end\n");
00214     #endif

```

```

00215     return 1;
00216
00217 exit_on_error:
00218     experiment_free (experiment, INPUT_TYPE_XML);
00219     #if DEBUG_EXPERIMENT
00220     fprintf (stderr, "experiment_open_xml: end\n");
00221     #endif
00222     return 0;
00223 }
00224
00225 int
00231 experiment_open_json (Experiment * experiment,
00232                      JsonNode * node,
00233                      unsigned int ninputs)
00234 {
00235     char buffer[64];
00236     JsonObject *object;
00237     const char *name;
00238     int error_code;
00239     unsigned int i;
00240
00241     #if DEBUG_EXPERIMENT
00242     fprintf (stderr, "experiment_open_json: start\n");
00243     #endif
00244
00245     // Resetting experiment data
00246     experiment_new (experiment);
00247
00248     // Getting JSON object
00249     object = json_node_get_object (node);
00250
00251     // Reading the experimental data
00252     name = json_object_get_string_member (object, LABEL_NAME);
00253     if (!name)
00254     {
00255         experiment_error (experiment, _("no data file name"));
00256         goto exit_on_error;
00257     }
00258     experiment->name = g_strdup (name);
00259     #if DEBUG_EXPERIMENT
00260     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00261     #endif
00262     experiment->weight
00263     = json_object_get_float_with_default (object,
00264     LABEL_WEIGHT, 1.,
00265     &error_code);
00266     if (error_code)
00267     {
00268         experiment_error (experiment, _("bad weight"));
00269         goto exit_on_error;
00270     }
00271     #if DEBUG_EXPERIMENT
00272     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00273     #endif
00274     name = json_object_get_string_member (object, stencil[0]);
00275     if (name)
00276     {
00277         #if DEBUG_EXPERIMENT
00278         fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00279         name, stencil[0]);
00280         #endif
00281         ++experiment->ninputs;
00282     }
00283     else
00284     {
00285         experiment_error (experiment, _("no template"));
00286         goto exit_on_error;
00287     }
00288     experiment->stencil[0] = g_strdup (name);
00289     for (i = 1; i < MAX_NINPUTS; ++i)
00290     {
00291         #if DEBUG_EXPERIMENT
00292         fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00293         #endif
00294         if (json_object_get_member (object, stencil[i]))
00295         {
00296             if (ninputs && ninputs <= i)
00297             {
00298                 experiment_error (experiment, _("bad templates number"));
00299                 goto exit_on_error;
00300             }
00301             name = json_object_get_string_member (object, stencil[i]);
00302             #if DEBUG_EXPERIMENT
00303             fprintf (stderr,
00304             "experiment_open_json: experiment=%s stencil%u=%s\n",
00305             experiment->nexperiments, name, stencil[i]);
00306             #endif

```



```

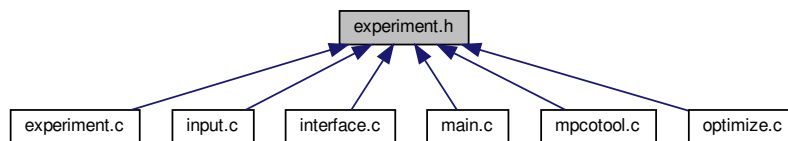
00307         experiment->stencil[i] = g_strdup (name);
00308         ++experiment->ninputs;
00309     }
00310     else if (ninputs && ninputs > i)
00311     {
00312         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00313         experiment_error (experiment, buffer);
00314         goto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

```

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_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.

4.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.5.2 Function Documentation

4.5.2.1 `experiment_error()`

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>message</i>	Error message.

Definition at line 109 of file [experiment.c](#).

```
00111 {  
00112     char buffer[64];  
00113     if (!experiment->name)  
00114         snprintf (buffer, 64, "%s: %s", _("Experiment"), message);  
00115     else  
00116         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),  
00117                     experiment->name, message);  
00118     error_message = g_strdup (buffer);  
00119 }
```

4.5.2.2 `experiment_free()`

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>type</i>	Type of input file.

Definition at line 80 of file [experiment.c](#).

```

00082 {
00083     unsigned int i;
00084     #if DEBUG_EXPERIMENT
00085         fprintf (stderr, "experiment_free: start\n");
00086     #endif
00087     if (type == INPUT_TYPE_XML)
00088     {
00089         for (i = 0; i < experiment->ninputs; ++i)
00090             xmlFree (experiment->stencil[i]);
00091         xmlFree (experiment->name);
00092     }
00093     else
00094     {
00095         for (i = 0; i < experiment->ninputs; ++i)
00096             g_free (experiment->stencil[i]);
00097         g_free (experiment->name);
00098     }
00099     experiment->ninputs = 0;
00100     #if DEBUG_EXPERIMENT
00101         fprintf (stderr, "experiment_free: end\n");
00102     #endif
00103 }

```

4.5.2.3 experiment_new()

```

void experiment_new (
    Experiment * experiment )

```

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

Parameters

<i>experiment</i>	Experiment struct.
-------------------	------------------------------------

Definition at line 61 of file [experiment.c](#).

```

00062 {
00063     unsigned int i;
00064     #if DEBUG_EXPERIMENT
00065         fprintf (stderr, "experiment_new: start\n");
00066     #endif
00067     experiment->name = NULL;
00068     experiment->ninputs = 0;
00069     for (i = 0; i < MAX_NINPUTS; ++i)
00070         experiment->stencil[i] = NULL;
00071     #if DEBUG_EXPERIMENT
00072         fprintf (stderr, "input_new: end\n");
00073     #endif
00074 }

```

4.5.2.4 experiment_open_json()

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

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

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 231 of file [experiment.c](#).

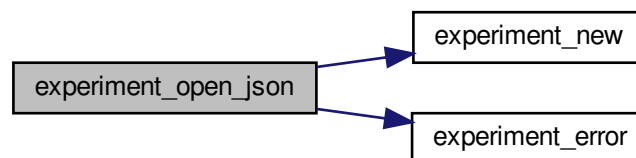
```
00235 {
00236     char buffer[64];
00237     JsonObject *object;
00238     const char *name;
00239     int error_code;
00240     unsigned int i;
00241
00242     #if DEBUG_EXPERIMENT
00243         fprintf (stderr, "experiment_open_json: start\n");
00244     #endif
00245
00246     // Resetting experiment data
00247     experiment_new (experiment);
00248
00249     // Getting JSON object
00250     object = json_node_get_object (node);
00251
00252     // Reading the experimental data
00253     name = json_object_get_string_member (object, LABEL_NAME);
00254     if (!name)
00255     {
00256         experiment_error (experiment, _("no data file name"));
00257         goto exit_on_error;
00258     }
00259     experiment->name = g_strdup (name);
00260     #if DEBUG_EXPERIMENT
00261         fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00262     #endif
00263     experiment->weight
00264         = json_object_get_float_with_default (object,
00265         LABEL_WEIGHT, 1.,
00266         &error_code);
00267     if (error_code)
00268     {
00269         experiment_error (experiment, _("bad weight"));
00270         goto exit_on_error;
00271     }
00272     #if DEBUG_EXPERIMENT
00273         fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00274     #endif
00275     name = json_object_get_string_member (object, stencil[0]);
00276     if (name)
00277     {
00278         #if DEBUG_EXPERIMENT
00279             fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00280                 name, stencil[0]);
00281         #endif
00282         ++experiment->ninputs;
00283     }
00284     else
```

```

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

```

Here is the call graph for this function:



4.5.2.5 experiment_open_xml()

```

int experiment_open_xml (
    Experiment * experiment,

```

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

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

Returns

1 on success, 0 on error.

Parameters

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

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

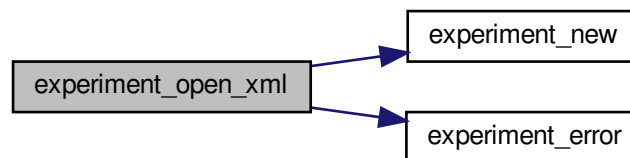
```
00131 {
00132     char buffer[64];
00133     int error_code;
00134     unsigned int i;
00135
00136     #if DEBUG_EXPERIMENT
00137         fprintf (stderr, "experiment_open_xml: start\n");
00138     #endif
00139
00140     // Resetting experiment data
00141     experiment_new (experiment);
00142
00143     // Reading the experimental data
00144     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145     if (!experiment->name)
00146     {
00147         experiment_error (experiment, _("no data file name"));
00148         goto exit_on_error;
00149     }
00150     #if DEBUG_EXPERIMENT
00151         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00152     #endif
00153     experiment->weight
00154         =
00155         xml_node_get_float_with_default (node, (const xmlChar *)
00156         LABEL_WEIGHT, 1.,
00157                                         &error_code);
00158     if (error_code)
00159     {
00160         experiment_error (experiment, _("bad weight"));
00161         goto exit_on_error;
00162     }
00163     #if DEBUG_EXPERIMENT
00164         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00165     #endif
00166     experiment->stencil[0]
00167         = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00168     if (experiment->stencil[0])
00169     {
00170         #if DEBUG_EXPERIMENT
00171             fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00172                     experiment->name, stencil[0]);
00173         #endif
00174         ++experiment->ninputs;
00175     }
00176     else
00177     {
00178         experiment_error (experiment, _("no template"));
00179         goto exit_on_error;
00180     }
00181     for (i = 1; i < MAX_NINPUTS; ++i)
00182     {
00183         #if DEBUG_EXPERIMENT
00184             fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00185         #endif
00186         if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00187         {
```

```

00187         if (ninputs && ninputs <= i)
00188         {
00189             experiment_error (experiment, _("bad templates number"));
00190             goto exit_on_error;
00191         }
00192         experiment->stencil[i]
00193         = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00194 #if DEBUG_EXPERIMENT
00195         fprintf (stderr,
00196                 "experiment_open_xml: experiment=%s stencil%u=%s\n",
00197                 experiment->nexperiments, experiment->name,
00198                 experiment->stencil[i]);
00199 #endif
00200         ++experiment->ninputs;
00201     }
00202     else if (ninputs && ninputs > i)
00203     {
00204         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00205         experiment_error (experiment, buffer);
00206         goto exit_on_error;
00207     }
00208     else
00209         break;
00210 }
00211
00212 #if DEBUG_EXPERIMENT
00213 fprintf (stderr, "experiment_open_xml: end\n");
00214 #endif
00215 return 1;
00216
00217 exit_on_error:
00218     experiment_free (experiment, INPUT_TYPE_XML);
00219 #if DEBUG_EXPERIMENT
00220     fprintf (stderr, "experiment_open_xml: end\n");
00221 #endif
00222     return 0;
00223 }

```

Here is the call graph for this function:



4.6 experiment.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013 1. Redistributions of source code must retain the above copyright notice,
00014    this list of conditions and the following disclaimer.
00015
00016 2. Redistributions in binary form must reproduce the above copyright notice,
00017    this list of conditions and the following disclaimer in the

```

```

00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1
00034
00035 typedef struct
00036 {
00037     char *name;
00038     char *stencil[MAX_NINPUTS];
00039     double weight;
00040     unsigned int ninputs;
00041 } Experiment;
00042
00043 extern const char *stencil[MAX_NINPUTS];
00044
00045 // Public functions
00046 void experiment_new (Experiment * experiment);
00047 void experiment_free (Experiment * experiment, unsigned int type);
00048 void experiment_error (Experiment * experiment, char *message);
00049 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00050                          unsigned int ninputs);
00051 int experiment_open_json (Experiment * experiment, JsonNode * node,
00052                           unsigned int ninputs);
00053
00054 #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 "utils.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](#) ()
- void [input_error](#) (char *message)
- int [input_open_xml](#) (xmlDoc *doc)
- int [input_open_json](#) (JsonParser *parser)
- int [input_open](#) (char *filename)

Variables

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

4.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.7.2 Function Documentation

4.7.2.1 [input_error\(\)](#)

```
void input_error (  
    char * message )
```

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

Parameters

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

Definition at line 119 of file [input.c](#).

```
00120 {
00121     char buffer[64];
00122     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00123     error_message = g_strdup (buffer);
00124 }
```

4.7.2.2 input_free()

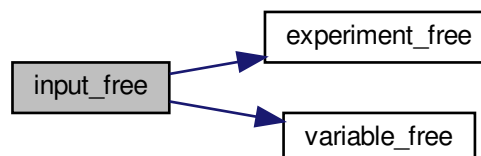
```
void input_free ( )
```

Function to free the memory of the input file data.

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

```
00082 {
00083     unsigned int i;
00084     #if DEBUG_INPUT
00085     fprintf (stderr, "input_free: start\n");
00086     #endif
00087     g_free (input->name);
00088     g_free (input->directory);
00089     for (i = 0; i < input->nexperiments; ++i)
00090         experiment_free (input->experiment + i, input->
type);
00091     for (i = 0; i < input->nvariables; ++i)
00092         variable_free (input->variable + i, input->
type);
00093     g_free (input->experiment);
00094     g_free (input->variable);
00095     if (input->type == INPUT_TYPE_XML)
00096     {
00097         xmlFree (input->evaluator);
00098         xmlFree (input->simulator);
00099         xmlFree (input->result);
00100         xmlFree (input->variables);
00101     }
00102     else
00103     {
00104         g_free (input->evaluator);
00105         g_free (input->simulator);
00106         g_free (input->result);
00107         g_free (input->variables);
00108     }
00109     input->nexperiments = input->nvariables =
input->nsteps = 0;
00110     #if DEBUG_INPUT
00111     fprintf (stderr, "input_free: end\n");
00112     #endif
00113 }
```

Here is the call graph for this function:



4.7.2.3 input_new()

```
void input_new ( )
```

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

Definition at line 63 of file [input.c](#).

```
00064 {
00065     #if DEBUG_INPUT
00066         fprintf (stderr, "input_new: start\n");
00067     #endif
00068     input->nvariables = input->nexperiments =
        input->nsteps = 0;
00069     input->simulator = input->evaluator = input->
        directory = input->name = NULL;
00070     input->experiment = NULL;
00071     input->variable = NULL;
00072     #if DEBUG_INPUT
00073         fprintf (stderr, "input_new: end\n");
00074     #endif
00075 }
```

4.7.2.4 input_open()

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

Function to open the input file.

Returns

1_on_success, 0_on_error.

Parameters

<i>filename</i>	Input data file name.
-----------------	---------------------------------------

Definition at line 957 of file [input.c](#).

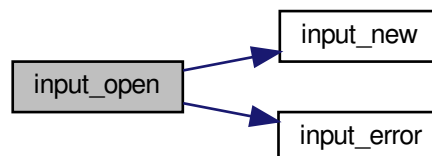
```
00958 {
00959     xmlDoc *doc;
00960     JsonParser *parser;
00961
00962     #if DEBUG_INPUT
00963         fprintf (stderr, "input_open: start\n");
00964     #endif
00965
00966     // Resetting input data
00967     input_new ();
00968
00969     // Opening input file
00970     #if DEBUG_INPUT
00971         fprintf (stderr, "input_open: opening the input file %s\n", filename);
00972         fprintf (stderr, "input_open: trying XML format\n");
00973     #endif
00974     doc = xmlParseFile (filename);
00975     if (!doc)
00976     {
00977         #if DEBUG_INPUT
```

```

00978     fprintf (stderr, "input_open: trying JSON format\n");
00979 #endif
00980     parser = json_parser_new ();
00981     if (!json_parser_load_from_file (parser, filename, NULL))
00982     {
00983         input_error (_("Unable to parse the input file"));
00984         goto exit_on_error;
00985     }
00986     if (!input_open_json (parser))
00987         goto exit_on_error;
00988 }
00989 else if (!input_open_xml (doc))
00990     goto exit_on_error;
00991
00992 // Getting the working directory
00993 input->directory = g_path_get_dirname (filename);
00994 input->name = g_path_get_basename (filename);
00995
00996 #if DEBUG_INPUT
00997     fprintf (stderr, "input_open: end\n");
00998 #endif
00999     return 1;
01000
01001 exit_on_error:
01002     show_error (error_message);
01003     g_free (error_message);
01004     input_free ();
01005 #if DEBUG_INPUT
01006     fprintf (stderr, "input_open: end\n");
01007 #endif
01008     return 0;
01009 }

```

Here is the call graph for this function:



4.7.2.5 input_open_json()

```

int input_open_json (
    JsonParser * parser )

```

Function to open the input file in JSON format.

Returns

1_on_success, 0_on_error.

Parameters

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

Definition at line 568 of file [input.c](#).

```

00569 {
00570     JsonNode *node, *child;
00571     JsonObject *object;
00572     JsonArray *array;
00573     const char *buffer;
00574     int error_code;
00575     unsigned int i, n;
00576
00577     #if DEBUG_INPUT
00578     fprintf (stderr, "input_open_json: start\n");
00579     #endif
00580
00581     // Resetting input data
00582     input->type = INPUT_TYPE_JSON;
00583
00584     // Getting the root node
00585     #if DEBUG_INPUT
00586     fprintf (stderr, "input_open_json: getting the root node\n");
00587     #endif
00588     node = json_parser_get_root (parser);
00589     object = json_node_get_object (node);
00590
00591     // Getting result and variables file names
00592     if (!input->result)
00593     {
00594         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00595         if (!buffer)
00596             buffer = result_name;
00597         input->result = g_strdup (buffer);
00598     }
00599     else
00600         input->result = g_strdup (result_name);
00601     if (!input->variables)
00602     {
00603         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00604         if (!buffer)
00605             buffer = variables_name;
00606         input->variables = g_strdup (buffer);
00607     }
00608     else
00609         input->variables = g_strdup (variables_name);
00610
00611     // Opening simulator program name
00612     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00613     if (!buffer)
00614     {
00615         input_error (_("Bad simulator program"));
00616         goto exit_on_error;
00617     }
00618     input->simulator = g_strdup (buffer);
00619
00620     // Opening evaluator program name
00621     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00622     if (buffer)
00623         input->evaluator = g_strdup (buffer);
00624
00625     // Obtaining pseudo-random numbers generator seed
00626     input->seed
00627     = json_object_get_uint_with_default (object,
00628     LABEL_SEED,
00629     DEFAULT_RANDOM_SEED, &error_code);
00630     if (error_code)
00631     {
00632         input_error (_("Bad pseudo-random numbers generator seed"));
00633         goto exit_on_error;
00634     }
00635
00636     // Opening algorithm
00637     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00638     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00639     {
00640         input->algorithm = ALGORITHM_MONTE_CARLO;
00641
00642         // Obtaining simulations number
00643         input->nsimulations
00644         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);
00645     }
00646     else if (!strcmp (buffer, LABEL_SWEEP))

```

```

00651     input->algorithm = ALGORITHM_SWEEP;
00652 else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00653     input->algorithm = ALGORITHM_ORTHOGONAL;
00654 else if (!strcmp (buffer, LABEL_GENETIC))
00655     {
00656         input->algorithm = ALGORITHM_GENETIC;
00657
00658         // Obtaining population
00659         if (json_object_get_member (object, LABEL_NPOPULATION))
00660         {
00661             input->nsimulations
00662             = json_object_get_uint (object,
00663 LABEL_NPOPULATION, &error_code);
00664             if (error_code || input->nsimulations < 3)
00665             {
00666                 input_error (_("Invalid population number"));
00667                 goto exit_on_error;
00668             }
00669         }
00670         else
00671         {
00672             input_error (_("No population number"));
00673             goto exit_on_error;
00674         }
00675
00676         // Obtaining generations
00677         if (json_object_get_member (object, LABEL_NGENERATIONS))
00678         {
00679             input->niterations
00680             = json_object_get_uint (object,
00681 LABEL_NGENERATIONS, &error_code);
00682             if (error_code || !input->niterations)
00683             {
00684                 input_error (_("Invalid generations number"));
00685                 goto exit_on_error;
00686             }
00687         }
00688         else
00689         {
00690             input_error (_("No generations number"));
00691             goto exit_on_error;
00692         }
00693
00694         // Obtaining mutation probability
00695         if (json_object_get_member (object, LABEL_MUTATION))
00696         {
00697             input->mutation_ratio
00698             = json_object_get_float (object, LABEL_MUTATION, &error_code
00699 );
00700             if (error_code || input->mutation_ratio < 0.
00701 || input->mutation_ratio >= 1.)
00702             {
00703                 input_error (_("Invalid mutation probability"));
00704                 goto exit_on_error;
00705             }
00706         }
00707         else
00708         {
00709             input_error (_("No mutation probability"));
00710             goto exit_on_error;
00711         }
00712
00713         // Obtaining reproduction probability
00714         if (json_object_get_member (object, LABEL_REPRODUCTION))
00715         {
00716             input->reproduction_ratio
00717             = json_object_get_float (object,
00718 LABEL_REPRODUCTION, &error_code);
00719             if (error_code || input->reproduction_ratio < 0.
00720 || input->reproduction_ratio >= 1.0)
00721             {
00722                 input_error (_("Invalid reproduction probability"));
00723                 goto exit_on_error;
00724             }
00725         }
00726         else
00727         {
00728             input_error (_("No reproduction probability"));
00729             goto exit_on_error;
00730         }
00731
00732         // Obtaining adaptation probability
00733         if (json_object_get_member (object, LABEL_ADAPTATION))
00734         {
00735             input->adaptation_ratio
00736             = json_object_get_float (object,
00737 LABEL_ADAPTATION, &error_code);

```

```

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

```

```

00815         input->nestimates
00816         = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00817         if (error_code || !input->nestimates)
00818         {
00819             input_error (_("Invalid estimates number"));
00820             goto exit_on_error;
00821         }
00822     }
00823     else
00824     {
00825         input_error (_("Unknown method to estimate the hill climbing"));
00826         goto exit_on_error;
00827     }
00828     input->relaxation
00829     = json_object_get_float_with_default (object,
LABEL_RELAXATION,
00830                                           DEFAULT_RELAXATION,
00831                                           &error_code);
00832     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00833     {
00834         input_error (_("Invalid relaxation parameter"));
00835         goto exit_on_error;
00836     }
00837 }
00838 else
00839     input->nsteps = 0;
00840 }
00841 // Obtaining the threshold
00842 input->threshold
00843 = json_object_get_float_with_default (object,
LABEL_THRESHOLD, 0.,
00844                                       &error_code);
00845 if (error_code)
00846 {
00847     input_error (_("Invalid threshold"));
00848     goto exit_on_error;
00849 }
00850
00851 // Reading the experimental data
00852 array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00853 n = json_array_get_length (array);
00854 input->experiment = (Experiment *) g_malloc (n * sizeof (
Experiment));
00855 for (i = 0; i < n; ++i)
00856 {
00857     #if DEBUG_INPUT
00858         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00859                 input->nexperiments);
00860     #endif
00861     child = json_array_get_element (array, i);
00862     if (!input->nexperiments)
00863     {
00864         if (!experiment_open_json (input->experiment, child, 0))
00865             goto exit_on_error;
00866     }
00867     else
00868     {
00869         if (!experiment_open_json (input->experiment +
input->nexperiments,
00870                                   child, input->experiment->
ninputs))
00871             goto exit_on_error;
00872     }
00873     ++input->nexperiments;
00874     #if DEBUG_INPUT
00875         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00876                 input->nexperiments);
00877     #endif
00878 }
00879 if (!input->nexperiments)
00880 {
00881     input_error (_("No optimization experiments"));
00882     goto exit_on_error;
00883 }
00884
00885 // Reading the variables data
00886 array = json_object_get_array_member (object, LABEL_VARIABLES);
00887 n = json_array_get_length (array);
00888 input->variable = (Variable *) g_malloc (n * sizeof (
Variable));
00889 for (i = 0; i < n; ++i)
00890 {
00891     #if DEBUG_INPUT
00892         fprintf (stderr, "input_open_json: nvariables=%u\n", input->
nvariables);

```



```

00893 #endif
00894     child = json_array_get_element (array, i);
00895     if (!variable_open_json (input->variable +
input->nvariables, child,
00896                             input->algorithm, input->
nsteps))
00897         goto exit_on_error;
00898     ++input->nvariables;
00899 }
00900 if (!input->nvariables)
00901 {
00902     input_error (_("No optimization variables"));
00903     goto exit_on_error;
00904 }
00905
00906 // Obtaining the error norm
00907 if (json_object_get_member (object, LABEL_NORM))
00908 {
00909     buffer = json_object_get_string_member (object, LABEL_NORM);
00910     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00911         input->norm = ERROR_NORM_EUCLIDIAN;
00912     else if (!strcmp (buffer, LABEL_MAXIMUM))
00913         input->norm = ERROR_NORM_MAXIMUM;
00914     else if (!strcmp (buffer, LABEL_P))
00915     {
00916         input->norm = ERROR_NORM_P;
00917         input->p = json_object_get_float (object,
LABEL_P, &error_code);
00918         if (!error_code)
00919         {
00920             input_error (_("Bad P parameter"));
00921             goto exit_on_error;
00922         }
00923     }
00924     else if (!strcmp (buffer, LABEL_TAXICAB))
00925         input->norm = ERROR_NORM_TAXICAB;
00926     else
00927     {
00928         input_error (_("Unknown error norm"));
00929         goto exit_on_error;
00930     }
00931 }
00932 else
00933     input->norm = ERROR_NORM_EUCLIDIAN;
00934
00935 // Closing the JSON document
00936 g_object_unref (parser);
00937
00938 #if DEBUG_INPUT
00939 fprintf (stderr, "input_open_json: end\n");
00940 #endif
00941 return 1;
00942
00943 exit_on_error:
00944 g_object_unref (parser);
00945 #if DEBUG_INPUT
00946 fprintf (stderr, "input_open_json: end\n");
00947 #endif
00948 return 0;
00949 }

```

Here is the call graph for this function:



4.7.2.6 input_open_xml()

```
int input_open_xml (
    xmlDoc * doc )
```

Function to open the input file in XML format.

Returns

1_on_success, 0_on_error.

Parameters

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

Definition at line 132 of file [input.c](#).

```
00133 {
00134     char buffer2[64];
00135     xmlNode *node, *child;
00136     xmlChar *buffer;
00137     int error_code;
00138     unsigned int i;
00139
00140     #if DEBUG_INPUT
00141         fprintf (stderr, "input_open_xml: start\n");
00142     #endif
00143
00144     // Resetting input data
00145     buffer = NULL;
00146     input->type = INPUT_TYPE_XML;
00147
00148     // Getting the root node
00149     #if DEBUG_INPUT
00150         fprintf (stderr, "input_open_xml: getting the root node\n");
00151     #endif
00152     node = xmlDocGetRootElement (doc);
00153     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00154     {
00155         input_error (_("Bad root XML node"));
00156         goto exit_on_error;
00157     }
00158
00159     // Getting result and variables file names
00160     if (!input->result)
00161     {
00162         input->result =
00163             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00164         if (!input->result)
00165             input->result = (char *) xmlStrdup ((const xmlChar *)
00166             result_name);
00167     }
00168     #if DEBUG_INPUT
00169         fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00170     #endif
00171     if (!input->variables)
00172     {
00173         input->variables =
00174             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00175         if (!input->variables)
00176             input->variables =
00177                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00178     }
00179     #if DEBUG_INPUT
00180         fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00181     #endif
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (_("Bad simulator program"));
00189         goto exit_on_error;
00190     }
00191 }
```

```

00189     }
00190
00191     // Opening evaluator program name
00192     input->evaluator =
00193         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00194
00195     // Obtaining pseudo-random numbers generator seed
00196     input->seed
00197         = xml_node_get_uint_with_default (node, (const xmlChar *)
00198     LABEL_SEED,
00199                                         DEFAULT_RANDOM_SEED, &error_code);
00199     if (error_code)
00200     {
00201         input_error (_("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }
00204
00205     // Opening algorithm
00206     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209         input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211         // Obtaining simulations number
00212         input->nsimulations
00213             = xml_node_get_int (node, (const xmlChar *)
00214     LABEL_NSIMULATIONS,
00215                             &error_code);
00216         if (error_code)
00217         {
00218             input_error (_("Bad simulations number"));
00219             goto exit_on_error;
00220         }
00221     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00222         input->algorithm = ALGORITHM_SWEEP;
00223     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00224         input->algorithm = ALGORITHM_ORTHOGONAL;
00225     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00226     {
00227         input->algorithm = ALGORITHM_GENETIC;
00228
00229         // Obtaining population
00230         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00231         {
00232             input->nsimulations
00233                 = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00234                                     &error_code);
00235             if (error_code || input->nsimulations < 3)
00236             {
00237                 input_error (_("Invalid population number"));
00238                 goto exit_on_error;
00239             }
00240         }
00241     else
00242     {
00243         input_error (_("No population number"));
00244         goto exit_on_error;
00245     }
00246
00247     // Obtaining generations
00248     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00249     {
00250         input->niterations
00251             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00252                                 &error_code);
00253         if (error_code || !input->niterations)
00254         {
00255             input_error (_("Invalid generations number"));
00256             goto exit_on_error;
00257         }
00258     }
00259     else
00260     {
00261         input_error (_("No generations number"));
00262         goto exit_on_error;
00263     }
00264
00265     // Obtaining mutation probability
00266     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00267     {
00268         input->mutation_ratio
00269             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00270                                  &error_code);
00271         if (error_code || input->mutation_ratio < 0.
00272             || input->mutation_ratio >= 1.)
00273         {

```

```

00274         input_error (_("Invalid mutation probability"));
00275         goto exit_on_error;
00276     }
00277 }
00278 else
00279 {
00280     input_error (_("No mutation probability"));
00281     goto exit_on_error;
00282 }
00283
00284 // Obtaining reproduction probability
00285 if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286 {
00287     input->reproduction_ratio
00288     = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00289                          &error_code);
00290     if (error_code || input->reproduction_ratio < 0.
00291         || input->reproduction_ratio >= 1.0)
00292     {
00293         input_error (_("Invalid reproduction probability"));
00294         goto exit_on_error;
00295     }
00296 }
00297 else
00298 {
00299     input_error (_("No reproduction probability"));
00300     goto exit_on_error;
00301 }
00302
00303 // Obtaining adaptation probability
00304 if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00305 {
00306     input->adaptation_ratio
00307     = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00308                          &error_code);
00309     if (error_code || input->adaptation_ratio < 0.
00310         || input->adaptation_ratio >= 1.)
00311     {
00312         input_error (_("Invalid adaptation probability"));
00313         goto exit_on_error;
00314     }
00315 }
00316 else
00317 {
00318     input_error (_("No adaptation probability"));
00319     goto exit_on_error;
00320 }
00321
00322 // Checking survivals
00323 i = input->mutation_ratio * input->nsimulations;
00324 i += input->reproduction_ratio * input->
00325 nsimulations;
00326 i += input->adaptation_ratio * input->
00327 nsimulations;
00328 if (i > input->nsimulations - 2)
00329 {
00330     input_error
00331     (_("No enough survival entities to reproduce the population"));
00332     goto exit_on_error;
00333 }
00334 else
00335 {
00336     input_error (_("Unknown algorithm"));
00337     goto exit_on_error;
00338 }
00339 xmlFree (buffer);
00340 buffer = NULL;
00341
00342 if (input->algorithm == ALGORITHM_MONTE_CARLO
00343     || input->algorithm == ALGORITHM_SWEEP
00344     || input->algorithm == ALGORITHM_ORTHOGONAL)
00345 {
00346     // Obtaining iterations number
00347     input->niterations
00348     = xml_node_get_uint (node, (const xmlChar *)
00349 LABEL_NITERATIONS,
00350                          &error_code);
00351     if (error_code == 1)
00352         input->niterations = 1;
00353     else if (error_code)
00354     {
00355         input_error (_("Bad iterations number"));
00356         goto exit_on_error;
00357     }

```

```

00358     // Obtaining best number
00359     input->nbest
00360     = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
00361                                     1, &error_code);
00362     if (error_code || !input->nbest)
00363     {
00364         input_error (_("Invalid best number"));
00365         goto exit_on_error;
00366     }
00367
00368     // Obtaining tolerance
00369     input->tolerance
00370     = xml_node_get_float_with_default (node,
00371                                     (const xmlChar *) LABEL_TOLERANCE,
00372                                     0., &error_code);
00373     if (error_code || input->tolerance < 0.)
00374     {
00375         input_error (_("Invalid tolerance"));
00376         goto exit_on_error;
00377     }
00378
00379     // Getting hill climbing method parameters
00380     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00381     {
00382         input->nsteps =
00383             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00384                               &error_code);
00385         if (error_code)
00386         {
00387             input_error (_("Invalid steps number"));
00388             goto exit_on_error;
00389         }
00390 #if DEBUG_INPUT
00391         fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00392 #endif
00393         buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00394         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->climbing = CLIMBING_METHOD_COORDINATES;
00396         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397         {
00398             input->climbing = CLIMBING_METHOD_RANDOM;
00399             input->nestimates
00400             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                                 &error_code);
00402             if (error_code || !input->nestimates)
00403             {
00404                 input_error (_("Invalid estimates number"));
00405                 goto exit_on_error;
00406             }
00407             else
00408             {
00409                 input_error (_("Unknown method to estimate the hill climbing"));
00410                 goto exit_on_error;
00411             }
00412             xmlFree (buffer);
00413             buffer = NULL;
00414             input->relaxation
00415             = xml_node_get_float_with_default (node,
00416                                               (const xmlChar *)
LABEL_RELAXATION,
00417                                               DEFAULT_RELAXATION, &error_code);
00420             if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00421             {
00422                 input_error (_("Invalid relaxation parameter"));
00423                 goto exit_on_error;
00424             }
00425             else
00426                 input->nsteps = 0;
00427         }
00428     }
00429     // Obtaining the threshold
00430     input->threshold =
00431         xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00432                                         0., &error_code);
00433     if (error_code)
00434     {
00435         input_error (_("Invalid threshold"));
00436         goto exit_on_error;
00437     }
00438
00439     // Reading the experimental data
00440     for (child = node->children; child; child = child->next)

```

```

00441     {
00442         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00443             break;
00444 #if DEBUG_INPUT
00445     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00446             input->nexperiments);
00447 #endif
00448     input->experiment = (Experiment *)
00449         g_realloc (input->experiment,
00450             (1 + input->nexperiments) * sizeof (
00451 Experiment));
00452     if (!input->nexperiments)
00453     {
00454         if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456     }
00457     else
00458     {
00459         if (!experiment_open_xml (input->experiment +
00460 input->nexperiments,
00461 child, input->experiment->
00462 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     if (!input->nexperiments)
00471     {
00472         input_error (_("No optimization experiments"));
00473         goto exit_on_error;
00474     }
00475     buffer = NULL;
00476     // Reading the variables data
00477     if (input->algorithm == ALGORITHM_SWEEP
00478         || input->algorithm == ALGORITHM_ORTHOGONAL)
00479         input->nsimulations = 1;
00480     for (; child; child = child->next)
00481     {
00482 #if DEBUG_INPUT
00483         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00484 #endif
00485         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00486         {
00487             snprintf (buffer2, 64, "%s %u: %s",
00488                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00489             input_error (buffer2);
00490             goto exit_on_error;
00491         }
00492         input->variable = (Variable *)
00493             g_realloc (input->variable,
00494                 (1 + input->nvariables) * sizeof (Variable));
00495         if (!variable_open_xml (input->variable +
00496 input->nvariables, child,
00497 input->algorithm, input->nsteps))
00498             goto exit_on_error;
00499         if (input->algorithm == ALGORITHM_SWEEP
00500             || input->algorithm == ALGORITHM_ORTHOGONAL)
00501             input->nsimulations *= input->variable[
00502 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     // Obtaining the error norm
00517     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00518     {
00519         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00520         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00521             input->norm = ERROR_NORM_EUCLIDIAN;
00522         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00523             input->norm = ERROR_NORM_MAXIMUM;
00524         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))

```

```

00523     {
00524         input->norm = ERROR_NORM_P;
00525         input->p
00526         = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00527         if (error_code)
00528         {
00529             input_error (_("Bad P parameter"));
00530             goto exit_on_error;
00531         }
00532     }
00533     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00534         input->norm = ERROR_NORM_TAXICAB;
00535     else
00536     {
00537         input_error (_("Unknown error norm"));
00538         goto exit_on_error;
00539     }
00540     xmlFree (buffer);
00541 }
00542 else
00543     input->norm = ERROR_NORM_EUCLIDIAN;
00544
00545 // Closing the XML document
00546 xmlFreeDoc (doc);
00547
00548 #if DEBUG_INPUT
00549 fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551 return 1;
00552
00553 exit_on_error:
00554     xmlFree (buffer);
00555     xmlFreeDoc (doc);
00556 #if DEBUG_INPUT
00557 fprintf (stderr, "input_open_xml: end\n");
00558 #endif
00559 return 0;
00560 }

```

Here is the call graph for this function:



4.8 input.c

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

```

```

00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <glib/gstdio.h>
00040 #include <json-glib/json-glib.h>
00041 #include "utils.h"
00042 #include "experiment.h"
00043 #include "variable.h"
00044 #include "input.h"
00045
00046 #define DEBUG_INPUT 0
00047
00048 Input input[1];
00049
00050 const char *result_name = "result";
00051 const char *variables_name = "variables";
00052
00053 void
00054 input_new ()
00055 {
00056     #if DEBUG_INPUT
00057         fprintf (stderr, "input_new: start\n");
00058     #endif
00059     input->nvariables = input->nexperiments = input->nsteps = 0;
00060     input->simulator = input->evaluator = input->directory = input->
name = NULL;
00061     input->experiment = NULL;
00062     input->variable = NULL;
00063     #if DEBUG_INPUT
00064         fprintf (stderr, "input_new: end\n");
00065     #endif
00066 }
00067
00068 void
00069 input_free ()
00070 {
00071     unsigned int i;
00072     #if DEBUG_INPUT
00073         fprintf (stderr, "input_free: start\n");
00074     #endif
00075     g_free (input->name);
00076     g_free (input->directory);
00077     for (i = 0; i < input->nexperiments; ++i)
00078         experiment_free (input->experiment + i, input->type);
00079     for (i = 0; i < input->nvariables; ++i)
00080         variable_free (input->variable + i, input->type);
00081     g_free (input->experiment);
00082     g_free (input->variable);
00083     if (input->type == INPUT_TYPE_XML)
00084     {
00085         xmlFree (input->evaluator);
00086         xmlFree (input->simulator);
00087         xmlFree (input->result);
00088         xmlFree (input->variables);
00089     }
00090     else
00091     {
00092         g_free (input->evaluator);
00093         g_free (input->simulator);
00094         g_free (input->result);
00095         g_free (input->variables);
00096     }
00097     input->nexperiments = input->nvariables = input->nsteps = 0;
00098     #if DEBUG_INPUT
00099         fprintf (stderr, "input_free: end\n");
00100     #endif
00101 }
00102
00103 void
00104 input_error (char *message)
00105 {
00106     char buffer[64];
00107     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);

```



```

00123     error_message = g_strdup (buffer);
00124 }
00125
00131 int
00132 input_open_xml (xmlDoc * doc)
00133 {
00134     char buffer2[64];
00135     xmlNode *node, *child;
00136     xmlChar *buffer;
00137     int error_code;
00138     unsigned int i;
00139
00140 #if DEBUG_INPUT
00141     fprintf (stderr, "input_open_xml: start\n");
00142 #endif
00143
00144     // Resetting input data
00145     buffer = NULL;
00146     input->type = INPUT_TYPE_XML;
00147
00148     // Getting the root node
00149 #if DEBUG_INPUT
00150     fprintf (stderr, "input_open_xml: getting the root node\n");
00151 #endif
00152     node = xmlDocGetRootElement (doc);
00153     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00154     {
00155         input_error (_("Bad root XML node"));
00156         goto exit_on_error;
00157     }
00158
00159     // Getting result and variables file names
00160     if (!input->result)
00161     {
00162         input->result =
00163             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00164         if (!input->result)
00165             input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00166     }
00167 #if DEBUG_INPUT
00168     fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00169 #endif
00170     if (!input->variables)
00171     {
00172         input->variables =
00173             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00174         if (!input->variables)
00175             input->variables =
00176                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00177     }
00178 #if DEBUG_INPUT
00179     fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00180 #endif
00181
00182     // Opening simulator program name
00183     input->simulator =
00184         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00185     if (!input->simulator)
00186     {
00187         input_error (_("Bad simulator program"));
00188         goto exit_on_error;
00189     }
00190
00191     // Opening evaluator program name
00192     input->evaluator =
00193         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00194
00195     // Obtaining pseudo-random numbers generator seed
00196     input->seed
00197         = xml_node_get_uint_with_default (node, (const xmlChar *)
00198 LABEL_SEED,
00199                                     DEFAULT_RANDOM_SEED, &error_code);
00199     if (error_code)
00200     {
00201         input_error (_("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }
00204
00205     // Opening algorithm
00206     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209         input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211         // Obtaining simulations number
00212         input->nsimulations
00213             = xml_node_get_int (node, (const xmlChar *)

```

```

    LABEL_NSIMULATIONS,
00214         &error_code);
00215     if (error_code)
00216     {
00217         input_error (_("Bad simulations number"));
00218         goto exit_on_error;
00219     }
00220 }
00221 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00222     input->algorithm = ALGORITHM_SWEEP;
00223 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00224     input->algorithm = ALGORITHM_ORTHOGONAL;
00225 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00226 {
00227     input->algorithm = ALGORITHM_GENETIC;
00228 }
00229 // Obtaining population
00230 if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00231 {
00232     input->nsimulations
00233     = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00234         &error_code);
00235     if (error_code || input->nsimulations < 3)
00236     {
00237         input_error (_("Invalid population number"));
00238         goto exit_on_error;
00239     }
00240 }
00241 else
00242 {
00243     input_error (_("No population number"));
00244     goto exit_on_error;
00245 }
00246 // Obtaining generations
00247 if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248 {
00249     input->niterations
00250     = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00251         &error_code);
00252     if (error_code || !input->niterations)
00253     {
00254         input_error (_("Invalid generations number"));
00255         goto exit_on_error;
00256     }
00257 }
00258 else
00259 {
00260     input_error (_("No generations number"));
00261     goto exit_on_error;
00262 }
00263 // Obtaining mutation probability
00264 if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00265 {
00266     input->mutation_ratio
00267     = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00268         &error_code);
00269     if (error_code || input->mutation_ratio < 0.
00270         || input->mutation_ratio >= 1.)
00271     {
00272         input_error (_("Invalid mutation probability"));
00273         goto exit_on_error;
00274     }
00275 }
00276 else
00277 {
00278     input_error (_("No mutation probability"));
00279     goto exit_on_error;
00280 }
00281 // Obtaining reproduction probability
00282 if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00283 {
00284     input->reproduction_ratio
00285     = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00286         &error_code);
00287     if (error_code || input->reproduction_ratio < 0.
00288         || input->reproduction_ratio >= 1.0)
00289     {
00290         input_error (_("Invalid reproduction probability"));
00291         goto exit_on_error;
00292     }
00293 }
00294 else
00295 {
00296     input_error (_("No reproduction probability"));
00297 }

```

```

00300         goto exit_on_error;
00301     }
00302
00303     // Obtaining adaptation probability
00304     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00305     {
00306         input->adaptation_ratio
00307         = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00308                               &error_code);
00309         if (error_code || input->adaptation_ratio < 0.
00310             || input->adaptation_ratio >= 1.)
00311         {
00312             input_error (_("Invalid adaptation probability"));
00313             goto exit_on_error;
00314         }
00315     }
00316     else
00317     {
00318         input_error (_("No adaptation probability"));
00319         goto exit_on_error;
00320     }
00321
00322     // Checking survivals
00323     i = input->mutation_ratio * input->nsimulations;
00324     i += input->reproduction_ratio * input->nsimulations;
00325     i += input->adaptation_ratio * input->nsimulations;
00326     if (i > input->nsimulations - 2)
00327     {
00328         input_error
00329         (_("No enough survival entities to reproduce the population"));
00330         goto exit_on_error;
00331     }
00332 }
00333 else
00334 {
00335     input_error (_("Unknown algorithm"));
00336     goto exit_on_error;
00337 }
00338 xmlFree (buffer);
00339 buffer = NULL;
00340
00341 if (input->algorithm == ALGORITHM_MONTE_CARLO
00342     || input->algorithm == ALGORITHM_SWEEP
00343     || input->algorithm == ALGORITHM_ORTHOGONAL)
00344 {
00345     // Obtaining iterations number
00346     input->niterations
00347     = xml_node_get_uint (node, (const xmlChar *)
00348 LABEL_NITERATIONS,
00349                         &error_code);
00350     if (error_code == 1)
00351         input->niterations = 1;
00352     else if (error_code)
00353     {
00354         input_error (_("Bad iterations number"));
00355         goto exit_on_error;
00356     }
00357
00358     // Obtaining best number
00359     input->nbest
00360     = xml_node_get_uint_with_default (node, (const xmlChar *)
00361 LABEL_NBEST,
00362                                     1, &error_code);
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     = xml_node_get_float_with_default (node,
00372                                       (const xmlChar *) LABEL_TOLERANCE,
00373                                       0., &error_code);
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             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00385                               &error_code);

```

```

00385         if (error_code)
00386         {
00387             input_error (_("Invalid steps number"));
00388             goto exit_on_error;
00389         }
00390 #if DEBUG_INPUT
00391         fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00392 #endif
00393         buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00394         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->climbing = CLIMBING_METHOD_COORDINATES;
00396         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397         {
00398             input->climbing = CLIMBING_METHOD_RANDOM;
00399             input->nestimates
00400             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                                 &error_code);
00402             if (error_code || !input->nestimates)
00403             {
00404                 input_error (_("Invalid estimates number"));
00405                 goto exit_on_error;
00406             }
00407         }
00408         else
00409         {
00410             input_error (_("Unknown method to estimate the hill climbing"));
00411             goto exit_on_error;
00412         }
00413         xmlFree (buffer);
00414         buffer = NULL;
00415         input->relaxation
00416         = xml_node_get_float_with_default (node,
00417                                           (const xmlChar *)
LABEL_RELAXATION,
00418                                           DEFAULT_RELAXATION, &error_code);
00419         if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00420         {
00421             input_error (_("Invalid relaxation parameter"));
00422             goto exit_on_error;
00423         }
00424     }
00425     else
00426         input->nsteps = 0;
00427 }
00428 // Obtaining the threshold
00429 input->threshold =
00430     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00431                                     0., &error_code);
00432 if (error_code)
00433 {
00434     input_error (_("Invalid threshold"));
00435     goto exit_on_error;
00436 }
00437 }
00438 // Reading the experimental data
00439 for (child = node->children; child; child = child->next)
00440 {
00441     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00442         break;
00443 #if DEBUG_INPUT
00444     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00445             input->nexperiments);
00446 #endif
00447     input->experiment = (Experiment *)
00448         g_realloc (input->experiment,
00449                   (1 + input->nexperiments) * sizeof (Experiment));
00450     if (!input->nexperiments)
00451     {
00452         if (!experiment_open_xml (input->experiment, child, 0))
00453             goto exit_on_error;
00454     }
00455     else
00456     {
00457         if (!experiment_open_xml (input->experiment + input->
nexperiments,
00458                                 child, input->experiment->ninputs))
00459             goto exit_on_error;
00460     }
00461     ++input->nexperiments;
00462 #if DEBUG_INPUT
00463     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00464             input->nexperiments);
00465 #endif
00466 }

```

```

00468     if (!input->nexperiments)
00469     {
00470         input_error (_("No optimization experiments"));
00471         goto exit_on_error;
00472     }
00473     buffer = NULL;
00474
00475     // Reading the variables data
00476     if (input->algorithm == ALGORITHM_SWEEP
00477         || input->algorithm == ALGORITHM_ORTHOGONAL)
00478         input->nsimulations = 1;
00479     for (; child; child = child->next)
00480     {
00481         #if DEBUG_INPUT
00482             fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00483         #endif
00484         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00485         {
00486             snprintf (buffer2, 64, "%s %u: %s",
00487                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00488             input_error (buffer2);
00489             goto exit_on_error;
00490         }
00491         input->variable = (Variable *)
00492             g_realloc (input->variable,
00493                 (1 + input->nvariables) * sizeof (Variable));
00494         if (!variable_open_xml (input->variable + input->
00495             nvariables, child,
00496                 input->algorithm, input->nsteps))
00497             goto exit_on_error;
00498         if (input->algorithm == ALGORITHM_SWEEP
00499             || input->algorithm == ALGORITHM_ORTHOGONAL)
00500             input->nsimulations *= input->variable[input->
00501             nvariables].nsweeps;
00502         ++input->nvariables;
00503     }
00504     if (!input->nvariables)
00505     {
00506         input_error (_("No optimization variables"));
00507         goto exit_on_error;
00508     }
00509     if (input->nbest > input->nsimulations)
00510     {
00511         input_error (_("Best number higher than simulations number"));
00512         goto exit_on_error;
00513     }
00514     buffer = NULL;
00515
00516     // Obtaining the error norm
00517     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00518     {
00519         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00520         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00521             input->norm = ERROR_NORM_EUCLIDIAN;
00522         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00523             input->norm = ERROR_NORM_MAXIMUM;
00524         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00525         {
00526             input->norm = ERROR_NORM_P;
00527             input->p
00528                 = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00529             if (error_code)
00530             {
00531                 input_error (_("Bad P parameter"));
00532                 goto exit_on_error;
00533             }
00534         }
00535         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00536             input->norm = ERROR_NORM_TAXICAB;
00537         else
00538         {
00539             input_error (_("Unknown error norm"));
00540             goto exit_on_error;
00541         }
00542         xmlFree (buffer);
00543     }
00544     else
00545         input->norm = ERROR_NORM_EUCLIDIAN;
00546
00547     // Closing the XML document
00548     xmlFreeDoc (doc);
00549
00550     #if DEBUG_INPUT
00551         fprintf (stderr, "input_open_xml: end\n");
00552     #endif
00553     return 1;
00554 }

```

```

00553 exit_on_error:
00554     xmlFree (buffer);
00555     xmlFreeDoc (doc);
00556 #if DEBUG_INPUT
00557     fprintf (stderr, "input_open_xml: end\n");
00558 #endif
00559     return 0;
00560 }
00561
00562 int
00563 input_open_json (JsonParser * parser)
00564 {
00565     JsonNode *node, *child;
00566     JsonObject *object;
00567     JsonArray *array;
00568     const char *buffer;
00569     int error_code;
00570     unsigned int i, n;
00571
00572 #if DEBUG_INPUT
00573     fprintf (stderr, "input_open_json: start\n");
00574 #endif
00575
00576     // Resetting input data
00577     input->type = INPUT_TYPE_JSON;
00578
00579     // Getting the root node
00580 #if DEBUG_INPUT
00581     fprintf (stderr, "input_open_json: getting the root node\n");
00582 #endif
00583     node = json_parser_get_root (parser);
00584     object = json_node_get_object (node);
00585
00586     // Getting result and variables file names
00587     if (!input->result)
00588     {
00589         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00590         if (!buffer)
00591             buffer = result_name;
00592         input->result = g_strdup (buffer);
00593     }
00594     else
00595         input->result = g_strdup (result_name);
00596     if (!input->variables)
00597     {
00598         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00599         if (!buffer)
00600             buffer = variables_name;
00601         input->variables = g_strdup (buffer);
00602     }
00603     else
00604         input->variables = g_strdup (variables_name);
00605
00606     // Opening simulator program name
00607     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00608     if (!buffer)
00609     {
00610         input_error (_("Bad simulator program"));
00611         goto exit_on_error;
00612     }
00613     input->simulator = g_strdup (buffer);
00614
00615     // Opening evaluator program name
00616     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00617     if (buffer)
00618         input->evaluator = g_strdup (buffer);
00619
00620     // Obtaining pseudo-random numbers generator seed
00621     input->seed
00622     = json_object_get_uint_with_default (object,
00623     LABEL_SEED,
00624     DEFAULT_RANDOM_SEED, &error_code);
00625     if (error_code)
00626     {
00627         input_error (_("Bad pseudo-random numbers generator seed"));
00628         goto exit_on_error;
00629     }
00630
00631     // Opening algorithm
00632     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00633     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00634     {
00635         input->algorithm = ALGORITHM_MONTE_CARLO;
00636     }
00637
00638     // Obtaining simulations number
00639     input->nsimulations
00640     = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);

```

```

    );
00644     if (error_code)
00645     {
00646         input_error (_("Bad simulations number"));
00647         goto exit_on_error;
00648     }
00649 }
00650 else if (!strcmp (buffer, LABEL_SWEEP))
00651     input->algorithm = ALGORITHM_SWEEP;
00652 else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00653     input->algorithm = ALGORITHM_ORTHOGONAL;
00654 else if (!strcmp (buffer, LABEL_GENETIC))
00655     {
00656         input->algorithm = ALGORITHM_GENETIC;
00657
00658         // Obtaining population
00659         if (json_object_get_member (object, LABEL_NPOPULATION))
00660         {
00661             input->nsimulations
00662             = json_object_get_uint (object,
00663 LABEL_NPOPULATION, &error_code);
00664             if (error_code || input->nsimulations < 3)
00665             {
00666                 input_error (_("Invalid population number"));
00667                 goto exit_on_error;
00668             }
00669         }
00670         else
00671         {
00672             input_error (_("No population number"));
00673             goto exit_on_error;
00674         }
00675         // Obtaining generations
00676         if (json_object_get_member (object, LABEL_NGENERATIONS))
00677         {
00678             input->niterations
00679             = json_object_get_uint (object,
00680 LABEL_NGENERATIONS, &error_code);
00681             if (error_code || !input->niterations)
00682             {
00683                 input_error (_("Invalid generations number"));
00684                 goto exit_on_error;
00685             }
00686         }
00687         else
00688         {
00689             input_error (_("No generations number"));
00690             goto exit_on_error;
00691         }
00692         // Obtaining mutation probability
00693         if (json_object_get_member (object, LABEL_MUTATION))
00694         {
00695             input->mutation_ratio
00696             = json_object_get_float (object, LABEL_MUTATION, &error_code
00697 );
00698             if (error_code || input->mutation_ratio < 0.
00699                 || input->mutation_ratio >= 1.)
00700             {
00701                 input_error (_("Invalid mutation probability"));
00702                 goto exit_on_error;
00703             }
00704         }
00705         else
00706         {
00707             input_error (_("No mutation probability"));
00708             goto exit_on_error;
00709         }
00710         // Obtaining reproduction probability
00711         if (json_object_get_member (object, LABEL_REPRODUCTION))
00712         {
00713             input->reproduction_ratio
00714             = json_object_get_float (object,
00715 LABEL_REPRODUCTION, &error_code);
00716             if (error_code || input->reproduction_ratio < 0.
00717                 || input->reproduction_ratio >= 1.0)
00718             {
00719                 input_error (_("Invalid reproduction probability"));
00720                 goto exit_on_error;
00721             }
00722         }
00723         else
00724         {
00725             input_error (_("No reproduction probability"));
00726             goto exit_on_error;

```

```

00726     }
00727
00728     // Obtaining adaptation probability
00729     if (json_object_get_member (object, LABEL_ADAPTATION))
00730     {
00731         input->adaptation_ratio
00732         = json_object_get_float (object,
00733 LABEL_ADAPTATION, &error_code);
00734         if (error_code || input->adaptation_ratio < 0.
00735             || input->adaptation_ratio >= 1.)
00736         {
00737             input_error (_("Invalid adaptation probability"));
00738             goto exit_on_error;
00739         }
00740     }
00741     else
00742     {
00743         input_error (_("No adaptation probability"));
00744         goto exit_on_error;
00745     }
00746     // Checking survivals
00747     i = input->mutation_ratio * input->nsimulations;
00748     i += input->reproduction_ratio * input->nsimulations;
00749     i += input->adaptation_ratio * input->nsimulations;
00750     if (i > input->nsimulations - 2)
00751     {
00752         input_error
00753         (_("No enough survival entities to reproduce the population"));
00754         goto exit_on_error;
00755     }
00756 }
00757 else
00758 {
00759     input_error (_("Unknown algorithm"));
00760     goto exit_on_error;
00761 }
00762
00763 if (input->algorithm == ALGORITHM_MONTE_CARLO
00764     || input->algorithm == ALGORITHM_SWEEP
00765     || input->algorithm == ALGORITHM_ORTHOGONAL)
00766 {
00767     // Obtaining iterations number
00768     input->niterations
00769     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00770 );
00771     if (error_code == 1)
00772         input->niterations = 1;
00773     else if (error_code)
00774     {
00775         input_error (_("Bad iterations number"));
00776         goto exit_on_error;
00777     }
00778     // Obtaining best number
00779     input->nbest
00780     = json_object_get_uint_with_default (object,
00781 LABEL_NBEST, 1,
00782                                         &error_code);
00783     if (error_code || !input->nbest)
00784     {
00785         input_error (_("Invalid best number"));
00786         goto exit_on_error;
00787     }
00788     // Obtaining tolerance
00789     input->tolerance
00790     = json_object_get_float_with_default (object,
00791 LABEL_TOLERANCE, 0.,
00792                                         &error_code);
00793     if (error_code || input->tolerance < 0.)
00794     {
00795         input_error (_("Invalid tolerance"));
00796         goto exit_on_error;
00797     }
00798     // Getting hill climbing method parameters
00799     if (json_object_get_member (object, LABEL_NSTEPS))
00800     {
00801         input->nsteps
00802         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00803         if (error_code)
00804         {
00805             input_error (_("Invalid steps number"));
00806             goto exit_on_error;
00807         }
00808     }

```



```

00809     buffer = json_object_get_string_member (object, LABEL_CLIMBING);
00810     if (!strcmp (buffer, LABEL_COORDINATES))
00811         input->climbing = CLIMBING_METHOD_COORDINATES;
00812     else if (!strcmp (buffer, LABEL_RANDOM))
00813     {
00814         input->climbing = CLIMBING_METHOD_RANDOM;
00815         input->nestimates
00816             = json_object_get_uint (object,
00817 LABEL_NESTIMATES, &error_code);
00818         if (error_code || !input->nestimates)
00819         {
00820             input_error (_("Invalid estimates number"));
00821             goto exit_on_error;
00822         }
00823     }
00824     else
00825     {
00826         input_error (_("Unknown method to estimate the hill climbing"));
00827         goto exit_on_error;
00828     }
00829     input->relaxation
00830         = json_object_get_float_with_default (object,
00831 LABEL_RELAXATION,
00832                                             DEFAULT_RELAXATION,
00833                                             &error_code);
00834     if (error_code || input->relaxation < 0. || input->
00835 relaxation > 2.)
00836     {
00837         input_error (_("Invalid relaxation parameter"));
00838         goto exit_on_error;
00839     }
00840     else
00841         input->nsteps = 0;
00842     // Obtaining the threshold
00843     input->threshold
00844         = json_object_get_float_with_default (object,
00845 LABEL_THRESHOLD, 0.,
00846                                             &error_code);
00847     if (error_code)
00848     {
00849         input_error (_("Invalid threshold"));
00850         goto exit_on_error;
00851     }
00852     // Reading the experimental data
00853     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00854     n = json_array_get_length (array);
00855     input->experiment = (Experiment *) g_malloc (n * sizeof (
00856 Experiment));
00857     for (i = 0; i < n; ++i)
00858     {
00859         #if DEBUG_INPUT
00860             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00861 input->nexperiments);
00862         #endif
00863         child = json_array_get_element (array, i);
00864         if (!input->nexperiments)
00865         {
00866             if (!experiment_open_json (input->experiment, child, 0))
00867                 goto exit_on_error;
00868         }
00869         else
00870         {
00871             if (!experiment_open_json (input->experiment + input->
00872 nexperiments,
00873                                     child, input->experiment->ninputs))
00874                 goto exit_on_error;
00875         }
00876         ++input->nexperiments;
00877     }
00878     #if DEBUG_INPUT
00879         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00880 input->nexperiments);
00881     #endif
00882     if (!input->nexperiments)
00883     {
00884         input_error (_("No optimization experiments"));
00885         goto exit_on_error;
00886     }
00887     // Reading the variables data
00888     array = json_object_get_array_member (object, LABEL_VARIABLES);
00889     n = json_array_get_length (array);
00890     input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00891     for (i = 0; i < n; ++i)

```

```

00890     {
00891     #if DEBUG_INPUT
00892         fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00893     #endif
00894         child = json_array_get_element (array, i);
00895         if (!variable_open_json (input->variable + input->
nvariables, child,
00896                                 input->algorithm, input->nsteps))
00897             goto exit_on_error;
00898         ++input->nvariables;
00899     }
00900     if (!input->nvariables)
00901     {
00902         input_error (_("No optimization variables"));
00903         goto exit_on_error;
00904     }
00905
00906     // Obtaining the error norm
00907     if (json_object_get_member (object, LABEL_NORM))
00908     {
00909         buffer = json_object_get_string_member (object, LABEL_NORM);
00910         if (!strcmp (buffer, LABEL_EUCLIDIAN))
00911             input->norm = ERROR_NORM_EUCLIDIAN;
00912         else if (!strcmp (buffer, LABEL_MAXIMUM))
00913             input->norm = ERROR_NORM_MAXIMUM;
00914         else if (!strcmp (buffer, LABEL_P))
00915         {
00916             input->norm = ERROR_NORM_P;
00917             input->p = json_object_get_float (object,
LABEL_P, &error_code);
00918             if (!error_code)
00919             {
00920                 input_error (_("Bad P parameter"));
00921                 goto exit_on_error;
00922             }
00923         }
00924         else if (!strcmp (buffer, LABEL_TAXICAB))
00925             input->norm = ERROR_NORM_TAXICAB;
00926         else
00927         {
00928             input_error (_("Unknown error norm"));
00929             goto exit_on_error;
00930         }
00931     }
00932     else
00933         input->norm = ERROR_NORM_EUCLIDIAN;
00934
00935     // Closing the JSON document
00936     g_object_unref (parser);
00937
00938     #if DEBUG_INPUT
00939     fprintf (stderr, "input_open_json: end\n");
00940     #endif
00941     return 1;
00942
00943 exit_on_error:
00944     g_object_unref (parser);
00945     #if DEBUG_INPUT
00946     fprintf (stderr, "input_open_json: end\n");
00947     #endif
00948     return 0;
00949 }
00950
00951 int
00952 input_open (char *filename)
00953 {
00954     xmlDoc *doc;
00955     JsonParser *parser;
00956
00957     #if DEBUG_INPUT
00958     fprintf (stderr, "input_open: start\n");
00959     #endif
00960
00961     // Resetting input data
00962     input_new ();
00963
00964     // Opening input file
00965     #if DEBUG_INPUT
00966     fprintf (stderr, "input_open: opening the input file %s\n", filename);
00967     fprintf (stderr, "input_open: trying XML format\n");
00968     #endif
00969     doc = xmlParseFile (filename);
00970     if (!doc)
00971     {
00972         #if DEBUG_INPUT
00973         fprintf (stderr, "input_open: trying JSON format\n");
00974         #endif
00975     }
00976 }

```

```

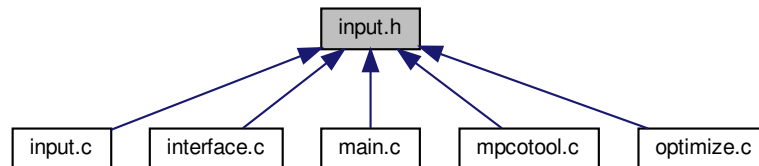
00980     parser = json_parser_new ();
00981     if (!json_parser_load_from_file (parser, filename, NULL))
00982     {
00983         input_error (_("Unable to parse the input file"));
00984         goto exit_on_error;
00985     }
00986     if (!input_open_json (parser))
00987         goto exit_on_error;
00988 }
00989 else if (!input_open_xml (doc))
00990     goto exit_on_error;
00991
00992 // Getting the working directory
00993 input->directory = g_path_get_dirname (filename);
00994 input->name = g_path_get_basename (filename);
00995
00996 #if DEBUG_INPUT
00997     fprintf (stderr, "input_open: end\n");
00998 #endif
00999     return 1;
01000
01001 exit_on_error:
01002     show_error (error_message);
01003     g_free (error_message);
01004     input_free ();
01005 #if DEBUG_INPUT
01006     fprintf (stderr, "input_open: end\n");
01007 #endif
01008     return 0;
01009 }

```

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](#) ()
- void [input_error](#) (char *message)
- int [input_open_xml](#) (xmlDoc *doc)
- 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](#)
Name of the result file.
- const char * [variables_name](#)
Name of the variables file.

4.9.1 Detailed Description

Header file to define the input functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.9.2 Enumeration Type Documentation

4.9.2.1 ClimbingMethod

enum [ClimbingMethod](#)

Enum to define the methods to estimate the hill climbing.

Enumerator

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

Definition at line 42 of file [input.h](#).

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

4.9.2.2 ErrorNorm

enum [ErrorNorm](#)

Enum to define the error norm.

Enumerator

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

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

```
00050 {
00051     ERROR_NORM_EUCLIDIAN = 0,
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_error()

```
void input_error (
    char * message )
```

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

Parameters

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

Definition at line 119 of file [input.c](#).

```

00120 {
00121     char buffer[64];
00122     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00123     error_message = g_strdup (buffer);
00124 }

```

4.9.3.2 input_free()

```
void input_free ( )
```

Function to free the memory of the input file data.

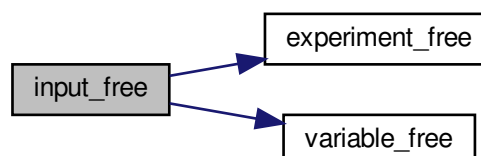
Definition at line 81 of file [input.c](#).

```

00082 {
00083     unsigned int i;
00084     #if DEBUG_INPUT
00085         fprintf (stderr, "input_free: start\n");
00086     #endif
00087     g_free (input->name);
00088     g_free (input->directory);
00089     for (i = 0; i < input->nexperiments; ++i)
00090         experiment_free (input->experiment + i, input->
type);
00091     for (i = 0; i < input->nvariables; ++i)
00092         variable_free (input->variable + i, input->
type);
00093     g_free (input->experiment);
00094     g_free (input->variable);
00095     if (input->type == INPUT_TYPE_XML)
00096     {
00097         xmlFree (input->evaluator);
00098         xmlFree (input->simulator);
00099         xmlFree (input->result);
00100         xmlFree (input->variables);
00101     }
00102     else
00103     {
00104         g_free (input->evaluator);
00105         g_free (input->simulator);
00106         g_free (input->result);
00107         g_free (input->variables);
00108     }
00109     input->nexperiments = input->nvariables =
input->nsteps = 0;
00110     #if DEBUG_INPUT
00111         fprintf (stderr, "input_free: end\n");
00112     #endif
00113 }

```

Here is the call graph for this function:



4.9.3.3 input_new()

```
void input_new ( )
```

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

Definition at line 63 of file [input.c](#).

```
00064 {
00065     #if DEBUG_INPUT
00066         fprintf (stderr, "input_new: start\n");
00067     #endif
00068     input->nvariables = input->nexperiments =
        input->nsteps = 0;
00069     input->simulator = input->evaluator = input->
        directory = input->name = NULL;
00070     input->experiment = NULL;
00071     input->variable = NULL;
00072     #if DEBUG_INPUT
00073         fprintf (stderr, "input_new: end\n");
00074     #endif
00075 }
```

4.9.3.4 input_open()

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

Function to open the input file.

Returns

1_on_success, 0_on_error.

Parameters

<i>filename</i>	Input data file name.
-----------------	---------------------------------------

Definition at line 957 of file [input.c](#).

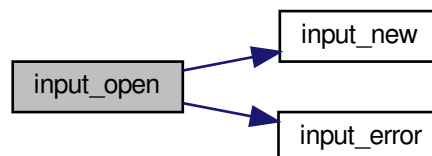
```
00958 {
00959     xmlDoc *doc;
00960     JsonParser *parser;
00961
00962     #if DEBUG_INPUT
00963         fprintf (stderr, "input_open: start\n");
00964     #endif
00965
00966     // Resetting input data
00967     input_new ();
00968
00969     // Opening input file
00970     #if DEBUG_INPUT
00971         fprintf (stderr, "input_open: opening the input file %s\n", filename);
00972         fprintf (stderr, "input_open: trying XML format\n");
00973     #endif
00974     doc = xmlParseFile (filename);
00975     if (!doc)
00976     {
00977         #if DEBUG_INPUT
```

```

00978     fprintf (stderr, "input_open: trying JSON format\n");
00979 #endif
00980     parser = json_parser_new ();
00981     if (!json_parser_load_from_file (parser, filename, NULL))
00982     {
00983         input_error (_("Unable to parse the input file"));
00984         goto exit_on_error;
00985     }
00986     if (!input_open_json (parser))
00987         goto exit_on_error;
00988 }
00989 else if (!input_open_xml (doc))
00990     goto exit_on_error;
00991
00992 // Getting the working directory
00993 input->directory = g_path_get_dirname (filename);
00994 input->name = g_path_get_basename (filename);
00995
00996 #if DEBUG_INPUT
00997     fprintf (stderr, "input_open: end\n");
00998 #endif
00999     return 1;
01000
01001 exit_on_error:
01002     show_error (error_message);
01003     g_free (error_message);
01004     input_free ();
01005 #if DEBUG_INPUT
01006     fprintf (stderr, "input_open: end\n");
01007 #endif
01008     return 0;
01009 }

```

Here is the call graph for this function:



4.9.3.5 input_open_json()

```

int input_open_json (
    JsonParser * parser )

```

Function to open the input file in JSON format.

Returns

1_on_success, 0_on_error.

Parameters

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

Definition at line 568 of file [input.c](#).

```

00569 {
00570     JsonNode *node, *child;
00571     JsonObject *object;
00572     JsonArray *array;
00573     const char *buffer;
00574     int error_code;
00575     unsigned int i, n;
00576
00577     #if DEBUG_INPUT
00578     fprintf (stderr, "input_open_json: start\n");
00579     #endif
00580
00581     // Resetting input data
00582     input->type = INPUT_TYPE_JSON;
00583
00584     // Getting the root node
00585     #if DEBUG_INPUT
00586     fprintf (stderr, "input_open_json: getting the root node\n");
00587     #endif
00588     node = json_parser_get_root (parser);
00589     object = json_node_get_object (node);
00590
00591     // Getting result and variables file names
00592     if (!input->result)
00593     {
00594         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00595         if (!buffer)
00596             buffer = result_name;
00597         input->result = g_strdup (buffer);
00598     }
00599     else
00600         input->result = g_strdup (result_name);
00601     if (!input->variables)
00602     {
00603         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00604         if (!buffer)
00605             buffer = variables_name;
00606         input->variables = g_strdup (buffer);
00607     }
00608     else
00609         input->variables = g_strdup (variables_name);
00610
00611     // Opening simulator program name
00612     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00613     if (!buffer)
00614     {
00615         input_error (_("Bad simulator program"));
00616         goto exit_on_error;
00617     }
00618     input->simulator = g_strdup (buffer);
00619
00620     // Opening evaluator program name
00621     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00622     if (buffer)
00623         input->evaluator = g_strdup (buffer);
00624
00625     // Obtaining pseudo-random numbers generator seed
00626     input->seed
00627     = json_object_get_uint_with_default (object,
00628     LABEL_SEED,
00629     DEFAULT_RANDOM_SEED, &error_code);
00630     if (error_code)
00631     {
00632         input_error (_("Bad pseudo-random numbers generator seed"));
00633         goto exit_on_error;
00634     }
00635
00636     // Opening algorithm
00637     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00638     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00639     {
00640         input->algorithm = ALGORITHM_MONTE_CARLO;
00641
00642         // Obtaining simulations number
00643         input->nsimulations
00644         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);
00645     }
00646     else if (!strcmp (buffer, LABEL_SWEEP))

```

```

00651     input->algorithm = ALGORITHM_SWEEP;
00652 else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00653     input->algorithm = ALGORITHM_ORTHOGONAL;
00654 else if (!strcmp (buffer, LABEL_GENETIC))
00655     {
00656         input->algorithm = ALGORITHM_GENETIC;
00657     }
00658     // Obtaining population
00659     if (json_object_get_member (object, LABEL_NPOPULATION))
00660     {
00661         input->nsimulations
00662             = json_object_get_uint (object,
00663 LABEL_NPOPULATION, &error_code);
00664         if (error_code || input->nsimulations < 3)
00665         {
00666             input_error (_("Invalid population number"));
00667             goto exit_on_error;
00668         }
00669     }
00670     else
00671     {
00672         input_error (_("No population number"));
00673         goto exit_on_error;
00674     }
00675     // Obtaining generations
00676     if (json_object_get_member (object, LABEL_NGENERATIONS))
00677     {
00678         input->niterations
00679             = json_object_get_uint (object,
00680 LABEL_NGENERATIONS, &error_code);
00681         if (error_code || !input->niterations)
00682         {
00683             input_error (_("Invalid generations number"));
00684             goto exit_on_error;
00685         }
00686     }
00687     else
00688     {
00689         input_error (_("No generations number"));
00690         goto exit_on_error;
00691     }
00692     // Obtaining mutation probability
00693     if (json_object_get_member (object, LABEL_MUTATION))
00694     {
00695         input->mutation_ratio
00696             = json_object_get_float (object, LABEL_MUTATION, &error_code
00697 );
00698         if (error_code || input->mutation_ratio < 0.
00699             || input->mutation_ratio >= 1.)
00700         {
00701             input_error (_("Invalid mutation probability"));
00702             goto exit_on_error;
00703         }
00704     }
00705     else
00706     {
00707         input_error (_("No mutation probability"));
00708         goto exit_on_error;
00709     }
00710     // Obtaining reproduction probability
00711     if (json_object_get_member (object, LABEL_REPRODUCTION))
00712     {
00713         input->reproduction_ratio
00714             = json_object_get_float (object,
00715 LABEL_REPRODUCTION, &error_code);
00716         if (error_code || input->reproduction_ratio < 0.
00717             || input->reproduction_ratio >= 1.0)
00718         {
00719             input_error (_("Invalid reproduction probability"));
00720             goto exit_on_error;
00721         }
00722     }
00723     else
00724     {
00725         input_error (_("No reproduction probability"));
00726         goto exit_on_error;
00727     }
00728     // Obtaining adaptation probability
00729     if (json_object_get_member (object, LABEL_ADAPTATION))
00730     {
00731         input->adaptation_ratio
00732             = json_object_get_float (object,
00733 LABEL_ADAPTATION, &error_code);

```

```

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

```

```

00815         input->nestimates
00816         = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00817         if (error_code || !input->nestimates)
00818         {
00819             input_error (_("Invalid estimates number"));
00820             goto exit_on_error;
00821         }
00822     }
00823     else
00824     {
00825         input_error (_("Unknown method to estimate the hill climbing"));
00826         goto exit_on_error;
00827     }
00828     input->relaxation
00829     = json_object_get_float_with_default (object,
LABEL_RELAXATION,
00830                                           DEFAULT_RELAXATION,
00831                                           &error_code);
00832     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00833     {
00834         input_error (_("Invalid relaxation parameter"));
00835         goto exit_on_error;
00836     }
00837 }
00838 else
00839     input->nsteps = 0;
00840 }
00841 // Obtaining the threshold
00842 input->threshold
00843 = json_object_get_float_with_default (object,
LABEL_THRESHOLD, 0.,
00844                                       &error_code);
00845 if (error_code)
00846 {
00847     input_error (_("Invalid threshold"));
00848     goto exit_on_error;
00849 }
00850
00851 // Reading the experimental data
00852 array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00853 n = json_array_get_length (array);
00854 input->experiment = (Experiment *) g_malloc (n * sizeof (
Experiment));
00855 for (i = 0; i < n; ++i)
00856 {
00857     #if DEBUG_INPUT
00858         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00859                 input->nexperiments);
00860     #endif
00861     child = json_array_get_element (array, i);
00862     if (!input->nexperiments)
00863     {
00864         if (!experiment_open_json (input->experiment, child, 0))
00865             goto exit_on_error;
00866     }
00867     else
00868     {
00869         if (!experiment_open_json (input->experiment +
input->nexperiments,
00870                                   child, input->experiment->
ninputs))
00871             goto exit_on_error;
00872     }
00873     ++input->nexperiments;
00874     #if DEBUG_INPUT
00875         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00876                 input->nexperiments);
00877     #endif
00878 }
00879 if (!input->nexperiments)
00880 {
00881     input_error (_("No optimization experiments"));
00882     goto exit_on_error;
00883 }
00884
00885 // Reading the variables data
00886 array = json_object_get_array_member (object, LABEL_VARIABLES);
00887 n = json_array_get_length (array);
00888 input->variable = (Variable *) g_malloc (n * sizeof (
Variable));
00889 for (i = 0; i < n; ++i)
00890 {
00891     #if DEBUG_INPUT
00892         fprintf (stderr, "input_open_json: nvariables=%u\n", input->
nvariables);

```

```

00893 #endif
00894     child = json_array_get_element (array, i);
00895     if (!variable_open_json (input->variable +
00896         input->nvariables, child,
00897         input->algorithm, input->
00898         nsteps))
00899         goto exit_on_error;
00900     ++input->nvariables;
00901 }
00902 if (!input->nvariables)
00903 {
00904     input_error (_("No optimization variables"));
00905     goto exit_on_error;
00906 }
00907 // Obtaining the error norm
00908 if (json_object_get_member (object, LABEL_NORM))
00909 {
00910     buffer = json_object_get_string_member (object, LABEL_NORM);
00911     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00912         input->norm = ERROR_NORM_EUCLIDIAN;
00913     else if (!strcmp (buffer, LABEL_MAXIMUM))
00914         input->norm = ERROR_NORM_MAXIMUM;
00915     else if (!strcmp (buffer, LABEL_P))
00916     {
00917         input->norm = ERROR_NORM_P;
00918         input->p = json_object_get_float (object,
00919             LABEL_P, &error_code);
00920         if (!error_code)
00921         {
00922             input_error (_("Bad P parameter"));
00923             goto exit_on_error;
00924         }
00925     }
00926     else if (!strcmp (buffer, LABEL_TAXICAB))
00927         input->norm = ERROR_NORM_TAXICAB;
00928     else
00929     {
00930         input_error (_("Unknown error norm"));
00931         goto exit_on_error;
00932     }
00933 }
00934 else
00935     input->norm = ERROR_NORM_EUCLIDIAN;
00936 // Closing the JSON document
00937 g_object_unref (parser);
00938 #if DEBUG_INPUT
00939 fprintf (stderr, "input_open_json: end\n");
00940 #endif
00941 return 1;
00942 }
00943 exit_on_error:
00944 g_object_unref (parser);
00945 #if DEBUG_INPUT
00946 fprintf (stderr, "input_open_json: end\n");
00947 #endif
00948 return 0;
00949 }

```

Here is the call graph for this function:



4.9.3.6 input_open_xml()

```
int input_open_xml (
    xmlDoc * doc )
```

Function to open the input file in XML format.

Returns

1_on_success, 0_on_error.

Parameters

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

Definition at line 132 of file [input.c](#).

```
00133 {
00134     char buffer2[64];
00135     xmlNode *node, *child;
00136     xmlChar *buffer;
00137     int error_code;
00138     unsigned int i;
00139
00140     #if DEBUG_INPUT
00141         fprintf (stderr, "input_open_xml: start\n");
00142     #endif
00143
00144     // Resetting input data
00145     buffer = NULL;
00146     input->type = INPUT_TYPE_XML;
00147
00148     // Getting the root node
00149     #if DEBUG_INPUT
00150         fprintf (stderr, "input_open_xml: getting the root node\n");
00151     #endif
00152     node = xmlDocGetRootElement (doc);
00153     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00154     {
00155         input_error (_("Bad root XML node"));
00156         goto exit_on_error;
00157     }
00158
00159     // Getting result and variables file names
00160     if (!input->result)
00161     {
00162         input->result =
00163             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00164         if (!input->result)
00165             input->result = (char *) xmlStrdup ((const xmlChar *)
00166             result_name);
00167     }
00168     #if DEBUG_INPUT
00169         fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00170     #endif
00171     if (!input->variables)
00172     {
00173         input->variables =
00174             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00175         if (!input->variables)
00176             input->variables =
00177                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00178     }
00179     #if DEBUG_INPUT
00180         fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00181     #endif
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (_("Bad simulator program"));
00189         goto exit_on_error;
00190     }
```

```

00189     }
00190
00191     // Opening evaluator program name
00192     input->evaluator =
00193         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00194
00195     // Obtaining pseudo-random numbers generator seed
00196     input->seed
00197         = xml_node_get_uint_with_default (node, (const xmlChar *)
00198         LABEL_SEED,
00199                                         DEFAULT_RANDOM_SEED, &error_code);
00199     if (error_code)
00200     {
00201         input_error (_("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }
00204
00205     // Opening algorithm
00206     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209         input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211         // Obtaining simulations number
00212         input->nsimulations
00213             = xml_node_get_int (node, (const xmlChar *)
00214             LABEL_NSIMULATIONS,
00215                               &error_code);
00216         if (error_code)
00217         {
00218             input_error (_("Bad simulations number"));
00219             goto exit_on_error;
00220         }
00221     }
00222     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223         input->algorithm = ALGORITHM_SWEEP;
00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00225         input->algorithm = ALGORITHM_ORTHOGONAL;
00226     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00227     {
00228         input->algorithm = ALGORITHM_GENETIC;
00229
00230         // Obtaining population
00231         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00232         {
00233             input->nsimulations
00234                 = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00235                                     &error_code);
00236             if (error_code || input->nsimulations < 3)
00237             {
00238                 input_error (_("Invalid population number"));
00239                 goto exit_on_error;
00240             }
00241         }
00242     }
00243     else
00244     {
00245         input_error (_("No population number"));
00246         goto exit_on_error;
00247     }
00248
00249     // Obtaining generations
00250     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00251     {
00252         input->niterations
00253             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00254                                 &error_code);
00255         if (error_code || !input->niterations)
00256         {
00257             input_error (_("Invalid generations number"));
00258             goto exit_on_error;
00259         }
00260     }
00261     else
00262     {
00263         input_error (_("No generations number"));
00264         goto exit_on_error;
00265     }
00266
00267     // Obtaining mutation probability
00268     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00269     {
00270         input->mutation_ratio
00271             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00272                                  &error_code);
00273         if (error_code || input->mutation_ratio < 0.
00274             || input->mutation_ratio >= 1.)
00275         {

```

```

00274         input_error (_("Invalid mutation probability"));
00275         goto exit_on_error;
00276     }
00277 }
00278 else
00279 {
00280     input_error (_("No mutation probability"));
00281     goto exit_on_error;
00282 }
00283
00284 // Obtaining reproduction probability
00285 if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286 {
00287     input->reproduction_ratio
00288     = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00289                          &error_code);
00290     if (error_code || input->reproduction_ratio < 0.
00291         || input->reproduction_ratio >= 1.0)
00292     {
00293         input_error (_("Invalid reproduction probability"));
00294         goto exit_on_error;
00295     }
00296 }
00297 else
00298 {
00299     input_error (_("No reproduction probability"));
00300     goto exit_on_error;
00301 }
00302
00303 // Obtaining adaptation probability
00304 if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00305 {
00306     input->adaptation_ratio
00307     = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00308                          &error_code);
00309     if (error_code || input->adaptation_ratio < 0.
00310         || input->adaptation_ratio >= 1.)
00311     {
00312         input_error (_("Invalid adaptation probability"));
00313         goto exit_on_error;
00314     }
00315 }
00316 else
00317 {
00318     input_error (_("No adaptation probability"));
00319     goto exit_on_error;
00320 }
00321
00322 // Checking survivals
00323 i = input->mutation_ratio * input->nsimulations;
00324 i += input->reproduction_ratio * input->
00325 nsimulations;
00326 i += input->adaptation_ratio * input->
00327 nsimulations;
00328 if (i > input->nsimulations - 2)
00329 {
00330     input_error
00331     (_("No enough survival entities to reproduce the population"));
00332     goto exit_on_error;
00333 }
00334 else
00335 {
00336     input_error (_("Unknown algorithm"));
00337     goto exit_on_error;
00338 }
00339 xmlFree (buffer);
00340 buffer = NULL;
00341
00342 if (input->algorithm == ALGORITHM_MONTE_CARLO
00343     || input->algorithm == ALGORITHM_SWEEP
00344     || input->algorithm == ALGORITHM_ORTHOGONAL)
00345 {
00346     // Obtaining iterations number
00347     input->niterations
00348     = xml_node_get_uint (node, (const xmlChar *)
00349 LABEL_NITERATIONS,
00350                          &error_code);
00351     if (error_code == 1)
00352         input->niterations = 1;
00353     else if (error_code)
00354     {
00355         input_error (_("Bad iterations number"));
00356         goto exit_on_error;
00357     }

```



```

00358     // Obtaining best number
00359     input->nbest
00360     = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
00361                                     1, &error_code);
00362     if (error_code || !input->nbest)
00363     {
00364         input_error (_("Invalid best number"));
00365         goto exit_on_error;
00366     }
00367
00368     // Obtaining tolerance
00369     input->tolerance
00370     = xml_node_get_float_with_default (node,
00371                                     (const xmlChar *) LABEL_TOLERANCE,
00372                                     0., &error_code);
00373     if (error_code || input->tolerance < 0.)
00374     {
00375         input_error (_("Invalid tolerance"));
00376         goto exit_on_error;
00377     }
00378
00379     // Getting hill climbing method parameters
00380     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00381     {
00382         input->nsteps =
00383             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00384                               &error_code);
00385         if (error_code)
00386         {
00387             input_error (_("Invalid steps number"));
00388             goto exit_on_error;
00389         }
00390 #if DEBUG_INPUT
00391         fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00392 #endif
00393         buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00394         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->climbing = CLIMBING_METHOD_COORDINATES;
00396         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397         {
00398             input->climbing = CLIMBING_METHOD_RANDOM;
00399             input->nestimates
00400             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                               &error_code);
00402             if (error_code || !input->nestimates)
00403             {
00404                 input_error (_("Invalid estimates number"));
00405                 goto exit_on_error;
00406             }
00407             else
00408             {
00409                 input_error (_("Unknown method to estimate the hill climbing"));
00410                 goto exit_on_error;
00411             }
00412             xmlFree (buffer);
00413             buffer = NULL;
00414             input->relaxation
00415             = xml_node_get_float_with_default (node,
00416                                               (const xmlChar *)
00417                                               LABEL_RELAXATION,
00418                                               DEFAULT_RELAXATION, &error_code);
00419             if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00420             {
00421                 input_error (_("Invalid relaxation parameter"));
00422                 goto exit_on_error;
00423             }
00424             else
00425             {
00426                 input->nsteps = 0;
00427             }
00428         }
00429     // Obtaining the threshold
00430     input->threshold =
00431         xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00432                                         0., &error_code);
00433     if (error_code)
00434     {
00435         input_error (_("Invalid threshold"));
00436         goto exit_on_error;
00437     }
00438
00439     // Reading the experimental data
00440     for (child = node->children; child; child = child->next)

```

```

00441     {
00442         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00443             break;
00444 #if DEBUG_INPUT
00445     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00446             input->nexperiments);
00447 #endif
00448     input->experiment = (Experiment *)
00449         g_realloc (input->experiment,
00450             (1 + input->nexperiments) * sizeof (
00451 Experiment));
00452     if (!input->nexperiments)
00453     {
00454         if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456     }
00457     else
00458     {
00459         if (!experiment_open_xml (input->experiment +
00460 input->nexperiments,
00461 child, input->experiment->
00462 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     if (!input->nexperiments)
00471     {
00472         input_error (_("No optimization experiments"));
00473         goto exit_on_error;
00474     }
00475     buffer = NULL;
00476     // Reading the variables data
00477     if (input->algorithm == ALGORITHM_SWEEP
00478         || input->algorithm == ALGORITHM_ORTHOGONAL)
00479         input->nsimulations = 1;
00480     for (; child; child = child->next)
00481     {
00482 #if DEBUG_INPUT
00483         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00484 #endif
00485         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00486         {
00487             snprintf (buffer2, 64, "%s %u: %s",
00488                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00489             input_error (buffer2);
00490             goto exit_on_error;
00491         }
00492         input->variable = (Variable *)
00493             g_realloc (input->variable,
00494                 (1 + input->nvariables) * sizeof (Variable));
00495         if (!variable_open_xml (input->variable +
00496 input->nvariables, child,
00497 input->algorithm, input->nsteps))
00498             goto exit_on_error;
00499         if (input->algorithm == ALGORITHM_SWEEP
00500             || input->algorithm == ALGORITHM_ORTHOGONAL)
00501             input->nsimulations *= input->variable[
00502 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     // Obtaining the error norm
00517     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00518     {
00519         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00520         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00521             input->norm = ERROR_NORM_EUCLIDIAN;
00522         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00523             input->norm = ERROR_NORM_MAXIMUM;
00524         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))

```

```

00523     {
00524         input->norm = ERROR_NORM_P;
00525         input->p
00526         = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00527         if (error_code)
00528         {
00529             input_error (_("Bad P parameter"));
00530             goto exit_on_error;
00531         }
00532     }
00533     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00534         input->norm = ERROR_NORM_TAXICAB;
00535     else
00536     {
00537         input_error (_("Unknown error norm"));
00538         goto exit_on_error;
00539     }
00540     xmlFree (buffer);
00541 }
00542 else
00543     input->norm = ERROR_NORM_EUCLIDIAN;
00544
00545 // Closing the XML document
00546 xmlFreeDoc (doc);
00547
00548 #if DEBUG_INPUT
00549 fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551 return 1;
00552
00553 exit_on_error:
00554     xmlFree (buffer);
00555     xmlFreeDoc (doc);
00556 #if DEBUG_INPUT
00557 fprintf (stderr, "input_open_xml: end\n");
00558 #endif
00559 return 0;
00560 }

```

Here is the call graph for this function:



4.10 input.h

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

```

```

00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INPUT__H
00033 #define INPUT__H 1
00034
00035 enum ClimbingMethod
00036 {
00037     CLIMBING_METHOD_COORDINATES = 0,
00038     CLIMBING_METHOD_RANDOM = 1,
00039 };
00040
00041 enum ErrorNorm
00042 {
00043     ERROR_NORM_EUCLIDIAN = 0,
00044     ERROR_NORM_MAXIMUM = 1,
00045     ERROR_NORM_P = 2,
00046     ERROR_NORM_TAXICAB = 3
00047 };
00048
00049 typedef struct
00050 {
00051     Experiment *experiment;
00052     Variable *variable;
00053     char *result;
00054     char *variables;
00055     char *simulator;
00056     char *evaluator;
00057     char *directory;
00058     char *name;
00059     double tolerance;
00060     double mutation_ratio;
00061     double reproduction_ratio;
00062     double adaptation_ratio;
00063     double relaxation;
00064     double p;
00065     double threshold;
00066     unsigned long int seed;
00067     unsigned int nvariables;
00068     unsigned int nexperiments;
00069     unsigned int nsimulations;
00070     unsigned int algorithm;
00071     unsigned int nsteps;
00072     unsigned int climbing;
00073     unsigned int nestimates;
00074     unsigned int niterations;
00075     unsigned int nbest;
00076     unsigned int norm;
00077     unsigned int type;
00078 } Input;
00079
00080 extern Input input[1];
00081 extern const char *result_name;
00082 extern const char *variables_name;
00083
00084 // Public functions
00085 void input_new ();
00086 void input_free ();
00087 void input_error (char *message);
00088 int input_open_xml (xmlDoc * doc);
00089 int input_open_json (JsonParser * parser);
00090 int input_open (char *filename);
00091
00092 #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 "genetic/genetic.h"
#include "utils.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](#) 0
Macro to debug interface functions.
- `#define` [INPUT_FILE](#) "test-ga.xml"
Macro to define the initial input file.

Functions

- void [input_save_climbing_xml](#) (xmlNode *node)
- void [input_save_climbing_json](#) (JsonNode *node)
- void [input_save_xml](#) (xmlDoc *doc)
- void [input_save_json](#) (JsonGenerator *generator)
- void [input_save](#) (char *filename)
- void [options_new](#) ()
- void [running_new](#) ()
- unsigned int [window_get_algorithm](#) ()
- unsigned int [window_get_climbing](#) ()
- unsigned int [window_get_norm](#) ()
- void [window_save_climbing](#) ()
- int [window_save](#) ()
- void [window_run](#) ()
- void [window_help](#) ()
- void [window_about](#) ()
- void [window_update_climbing](#) ()
- void [window_update](#) ()
- void [window_set_algorithm](#) ()
- void [window_set_experiment](#) ()
- void [window_remove_experiment](#) ()

- void [window_add_experiment](#) ()
- void [window_name_experiment](#) ()
- void [window_weight_experiment](#) ()
- void [window_inputs_experiment](#) ()
- void [window_template_experiment](#) (void *data)
- void [window_set_variable](#) ()
- void [window_remove_variable](#) ()
- void [window_add_variable](#) ()
- void [window_label_variable](#) ()
- void [window_precision_variable](#) ()
- void [window_rangemin_variable](#) ()
- void [window_rangemax_variable](#) ()
- void [window_rangeminabs_variable](#) ()
- void [window_rangemaxabs_variable](#) ()
- void [window_step_variable](#) ()
- void [window_update_variable](#) ()
- int [window_read](#) (char *filename)
- void [window_open](#) ()
- void [window_new](#) (GtkApplication *application)

Variables

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

4.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.11.2 Function Documentation

4.11.2.1 [input_save\(\)](#)

```
void input_save (
    char * filename )
```

Function to save the input file.

Parameters

<i>filename</i>	Input file name.
-----------------	------------------

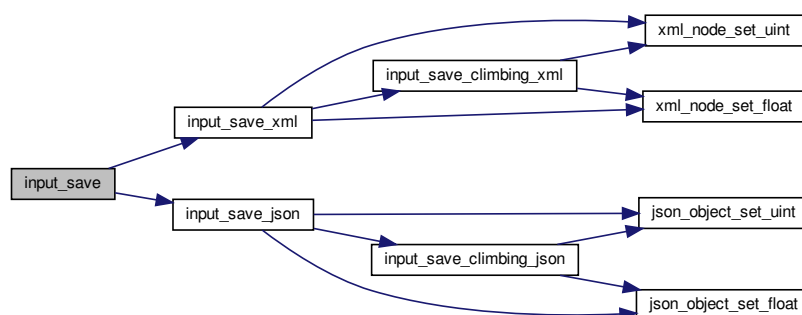
Definition at line 584 of file [interface.c](#).

```

00585 {
00586     xmlDoc *doc;
00587     JsonGenerator *generator;
00588
00589 #if DEBUG_INTERFACE
00590     fprintf (stderr, "input_save: start\n");
00591 #endif
00592
00593     // Getting the input file directory
00594     input->name = g_path_get_basename (filename);
00595     input->directory = g_path_get_dirname (filename);
00596
00597     if (input->type == INPUT_TYPE_XML)
00598     {
00599         // Opening the input file
00600         doc = xmlNewDoc ((const xmlChar *) "1.0");
00601         input_save_xml (doc);
00602
00603         // Saving the XML file
00604         xmlSaveFormatFile (filename, doc, 1);
00605
00606         // Freeing memory
00607         xmlFreeDoc (doc);
00608     }
00609     else
00610     {
00611         // Opening the input file
00612         generator = json_generator_new ();
00613         json_generator_set_pretty (generator, TRUE);
00614         input_save_json (generator);
00615
00616         // Saving the JSON file
00617         json_generator_to_file (generator, filename, NULL);
00618
00619         // Freeing memory
00620         g_object_unref (generator);
00621     }
00622
00623 #if DEBUG_INTERFACE
00624     fprintf (stderr, "input_save: end\n");
00625 #endif
00626 }

```

Here is the call graph for this function:



4.11.2.2 input_save_climbing_json()

```
void input_save_climbing_json (
    JsonNode * node )
```

Function to save the hill climbing method data in a JSON node.

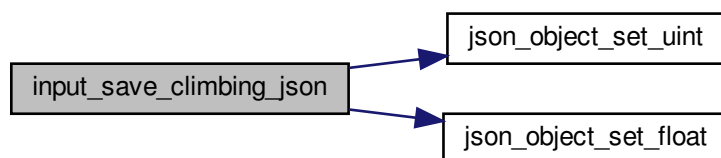
Parameters

<i>node</i>	JSON node.
-------------	------------

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

```
00202 {
00203     JsonObject *object;
00204     #if DEBUG_INTERFACE
00205     fprintf (stderr, "input_save_climbing_json: start\n");
00206     #endif
00207     object = json_node_get_object (node);
00208     if (input->nsteps)
00209     {
00210         json_object_set_uint (object, LABEL_NSTEPS,
00211             input->nsteps);
00212         if (input->relaxation != DEFAULT_RELAXATION)
00213             json_object_set_float (object, LABEL_RELAXATION,
00214                 input->relaxation);
00215         switch (input->climbing)
00216         {
00217             case CLIMBING_METHOD_COORDINATES:
00218                 json_object_set_string_member (object, LABEL_CLIMBING,
00219                     LABEL_COORDINATES);
00220                 break;
00221             default:
00222                 json_object_set_string_member (object, LABEL_CLIMBING,
00223                     LABEL_RANDOM);
00224         }
00225         json_object_set_uint (object, LABEL_NESTIMATES,
00226             input->nestimates);
00227     }
00228     #if DEBUG_INTERFACE
00229     fprintf (stderr, "input_save_climbing_json: end\n");
00230     #endif
00231 }
```

Here is the call graph for this function:



4.11.2.3 input_save_climbing_xml()

```
void input_save_climbing_xml (
    xmlNode * node )
```

Function to save the hill climbing method data in a XML node.

Parameters

<i>node</i>	XML node.
-------------	-----------

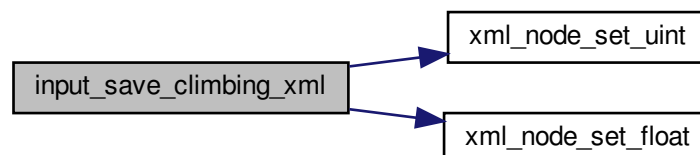
Definition at line 168 of file [interface.c](#).

```

00169 {
00170     #if DEBUG_INTERFACE
00171         fprintf (stderr, "input_save_climbing_xml: start\n");
00172     #endif
00173     if (input->nsteps)
00174     {
00175         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00176             input->nsteps);
00177         if (input->relaxation != DEFAULT_RELAXATION)
00178             xml_node_set_float (node, (const xmlChar *)
00179                 LABEL_RELAXATION,
00180                 input->relaxation);
00181         switch (input->climbing)
00182         {
00183             case CLIMBING_METHOD_COORDINATES:
00184                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00185                     (const xmlChar *) LABEL_COORDINATES);
00186                 break;
00187             default:
00188                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00189                     (const xmlChar *) LABEL_RANDOM);
00190                 xml_node_set_uint (node, (const xmlChar *)
00191                     LABEL_NESTIMATES,
00192                     input->nestimates);
00193         }
00194     }
00195     #if DEBUG_INTERFACE
00196         fprintf (stderr, "input_save_climbing_xml: end\n");
00197     #endif
00198 }

```

Here is the call graph for this function:



4.11.2.4 input_save_json()

```

void input_save_json (
    JsonGenerator * generator )

```

Function to save the input file in JSON format.

Parameters

<i>generator</i>	JsonGenerator struct.
------------------	-----------------------

Definition at line 413 of file [interface.c](#).

```

00414 {
00415     unsigned int i, j;
00416     char *buffer;
00417     JsonNode *node, *child;
00418     JsonObject *object;
00419     JsonArray *array;
00420     GFile *file, *file2;
00421
00422     #if DEBUG_INTERFACE
00423         fprintf (stderr, "input_save_json: start\n");
00424     #endif
00425
00426     // Setting root JSON node
00427     node = json_node_new (JSON_NODE_OBJECT);
00428     object = json_node_get_object (node);
00429     json_generator_set_root (generator, node);
00430
00431     // Adding properties to the root JSON node
00432     if (strcmp (input->result, result_name))
00433         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00434     if (strcmp (input->variables, variables_name))
00435         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00436     file = g_file_new_for_path (input->directory);
00437     file2 = g_file_new_for_path (input->simulator);
00438     buffer = g_file_get_relative_path (file, file2);
00439     g_object_unref (file2);
00440     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00441     g_free (buffer);
00442     if (input->evaluator)
00443     {
00444         file2 = g_file_new_for_path (input->evaluator);
00445         buffer = g_file_get_relative_path (file, file2);
00446         g_object_unref (file2);
00447         if (strlen (buffer))
00448             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00449         g_free (buffer);
00450     }
00451     if (input->seed != DEFAULT_RANDOM_SEED)
00452         json_object_set_uint (object, LABEL_SEED,
input->seed);
00453
00454     // Setting the algorithm
00455     buffer = (char *) g_slice_alloc (64);
00456     switch (input->algorithm)
00457     {
00458     case ALGORITHM_MONTE_CARLO:
00459         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00460         snprintf (buffer, 64, "%u", input->nsimulations);
00461         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00462         snprintf (buffer, 64, "%u", input->niterations);
00463         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00464         snprintf (buffer, 64, "%.3lg", input->tolerance);
00465         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00466         snprintf (buffer, 64, "%u", input->nbest);
00467         json_object_set_string_member (object, LABEL_NBEST, buffer);
00468         input_save_climbing_json (node);
00469         break;
00470     case ALGORITHM_SWEEP:
00471         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00472         snprintf (buffer, 64, "%u", input->niterations);
00473         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00474         snprintf (buffer, 64, "%.3lg", input->tolerance);
00475         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00476         snprintf (buffer, 64, "%u", input->nbest);
00477         json_object_set_string_member (object, LABEL_NBEST, buffer);
00478         input_save_climbing_json (node);
00479         break;
00480     case ALGORITHM_ORTHOGONAL:
00481         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_ORTHOGONAL);
00482         snprintf (buffer, 64, "%u", input->niterations);
00483         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);

```

```

00486     snprintf (buffer, 64, "%.3lg", input->tolerance);
00487     json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00488     snprintf (buffer, 64, "%u", input->nbest);
00489     json_object_set_string_member (object, LABEL_NBEST, buffer);
00490     input_save_climbing_json (node);
00491     break;
00492     default:
00493         json_object_set_string_member (object, LABEL_ALGORITHM,
00494 LABEL_GENETIC);
00495         snprintf (buffer, 64, "%u", input->nsimulations);
00496         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00497         snprintf (buffer, 64, "%u", input->niterations);
00498         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00499         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00500         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00501         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00502         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00503         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00504         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00505         break;
00506     }
00507     g_slice_free1 (64, buffer);
00508     if (input->threshold != 0.)
00509         json_object_set_float (object, LABEL_THRESHOLD,
00510 input->threshold);
00511     // Setting the experimental data
00512     array = json_array_new ();
00513     for (i = 0; i < input->nexperiments; ++i)
00514     {
00515         child = json_node_new (JSON_NODE_OBJECT);
00516         object = json_node_get_object (child);
00517         json_object_set_string_member (object, LABEL_NAME,
00518 input->experiment[i].name);
00519         if (input->experiment[i].weight != 1.)
00520             json_object_set_float (object, LABEL_WEIGHT,
00521 input->experiment[i].weight);
00522         for (j = 0; j < input->experiment->ninputs; ++j)
00523             json_object_set_string_member (object, stencil[j],
00524 input->experiment[i].
00525 stencil[j]);
00526         json_array_add_element (array, child);
00527     }
00528     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00529     // Setting the variables data
00530     array = json_array_new ();
00531     for (i = 0; i < input->nvariables; ++i)
00532     {
00533         child = json_node_new (JSON_NODE_OBJECT);
00534         object = json_node_get_object (child);
00535         json_object_set_string_member (object, LABEL_NAME,
00536 input->variable[i].name);
00537         json_object_set_float (object, LABEL_MINIMUM,
00538 input->variable[i].rangemin);
00539         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00540             json_object_set_float (object,
00541 LABEL_ABSOLUTE_MINIMUM,
00542 input->variable[i].rangeminabs);
00543         json_object_set_float (object, LABEL_MAXIMUM,
00544 input->variable[i].rangemax);
00545         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00546             json_object_set_float (object,
00547 LABEL_ABSOLUTE_MAXIMUM,
00548 input->variable[i].rangemaxabs);
00549         if (input->variable[i].precision !=
00550 DEFAULT_PRECISION)
00551             json_object_set_uint (object, LABEL_PRECISION,
00552 input->variable[i].precision);
00553         if (input->algorithm == ALGORITHM_SWEEP
00554             || input->algorithm == ALGORITHM_ORTHOGONAL)
00555             json_object_set_uint (object, LABEL_NSWEEPS,
00556 input->variable[i].nsweeps);
00557         else if (input->algorithm == ALGORITHM_GENETIC)
00558             json_object_set_uint (object, LABEL_NBITS,
00559 input->variable[i].nbits);
00560         if (input->nsteps)
00561             json_object_set_float (object, LABEL_STEP,
00562 input->variable[i].step);
00563         json_array_add_element (array, child);
00564     }
00565     json_object_set_array_member (object, LABEL_VARIABLES, array);
00566     // Saving the error norm
00567     switch (input->norm)
00568     {
00569     case ERROR_NORM_MAXIMUM:

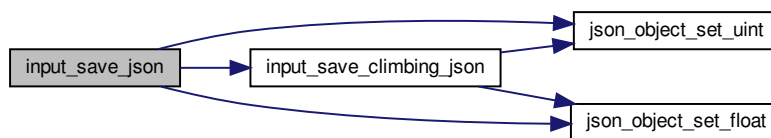
```

```

00565     json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00566     break;
00567 case ERROR_NORM_P:
00568     json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00569     json_object_set_float (object, LABEL_P, input->
p);
00570     break;
00571 case ERROR_NORM_TAXICAB:
00572     json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00573     }
00574
00575 #if DEBUG_INTERFACE
00576     fprintf (stderr, "input_save_json: end\n");
00577 #endif
00578 }

```

Here is the call graph for this function:



4.11.2.5 input_save_xml()

```

void input_save_xml (
    xmlDoc * doc )

```

Function to save the input file in XML format.

Parameters

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

Definition at line 233 of file [interface.c](#).

```

00234 {
00235     unsigned int i, j;
00236     char *buffer;
00237     xmlNode *node, *child;
00238     GFile *file, *file2;
00239
00240 #if DEBUG_INTERFACE
00241     fprintf (stderr, "input_save_xml: start\n");
00242 #endif
00243
00244     // Setting root XML node
00245     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00246     xmlDocSetRootElement (doc, node);
00247
00248     // Adding properties to the root XML node
00249     if (xmlStrcmp
00250         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00251         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00252             (xmlChar *) input->result);
00253     if (xmlStrcmp

```

```

00254     ((const xmlChar *) input->variables, (const xmlChar *)
variables_name))
00255     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00256                 (xmlChar *) input->variables);
00257     file = g_file_new_for_path (input->directory);
00258     file2 = g_file_new_for_path (input->simulator);
00259     buffer = g_file_get_relative_path (file, file2);
00260     g_object_unref (file2);
00261     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00262     g_free (buffer);
00263     if (input->evaluator)
00264     {
00265         file2 = g_file_new_for_path (input->evaluator);
00266         buffer = g_file_get_relative_path (file, file2);
00267         g_object_unref (file2);
00268         if (xmlStrlen ((xmlChar *) buffer))
00269             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00270                         (xmlChar *) buffer);
00271         g_free (buffer);
00272     }
00273     if (input->seed != DEFAULT_RANDOM_SEED)
00274         xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);
00275
00276     // Setting the algorithm
00277     buffer = (char *) g_slice_alloc (64);
00278     switch (input->algorithm)
00279     {
00280     case ALGORITHM_MONTE_CARLO:
00281         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00282                     (const xmlChar *) LABEL_MONTE_CARLO);
00283         snprintf (buffer, 64, "%u", input->nsimulations);
00284         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00285                     (xmlChar *) buffer);
00286         snprintf (buffer, 64, "%u", input->niterations);
00287         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00288                     (xmlChar *) buffer);
00289         snprintf (buffer, 64, "%.3lg", input->tolerance);
00290         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00291         snprintf (buffer, 64, "%u", input->nbest);
00292         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00293         input_save_climbing_xml (node);
00294         break;
00295     case ALGORITHM_SWEEP:
00296         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00297                     (const xmlChar *) LABEL_SWEEP);
00298         snprintf (buffer, 64, "%u", input->niterations);
00299         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00300                     (xmlChar *) buffer);
00301         snprintf (buffer, 64, "%.3lg", input->tolerance);
00302         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00303         snprintf (buffer, 64, "%u", input->nbest);
00304         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00305         input_save_climbing_xml (node);
00306         break;
00307     case ALGORITHM_ORTHOGONAL:
00308         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00309                     (const xmlChar *) LABEL_ORTHOGONAL);
00310         snprintf (buffer, 64, "%u", input->niterations);
00311         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00312                     (xmlChar *) buffer);
00313         snprintf (buffer, 64, "%.3lg", input->tolerance);
00314         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00315         snprintf (buffer, 64, "%u", input->nbest);
00316         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00317         input_save_climbing_xml (node);
00318         break;
00319     default:
00320         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00321                     (const xmlChar *) LABEL_GENETIC);
00322         snprintf (buffer, 64, "%u", input->nsimulations);
00323         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00324                     (xmlChar *) buffer);
00325         snprintf (buffer, 64, "%u", input->niterations);
00326         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00327                     (xmlChar *) buffer);
00328         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00329         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00332                     (xmlChar *) buffer);
00333         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00335         break;
00336     }
00337     g_slice_free1 (64, buffer);
00338     if (input->threshold != 0.)

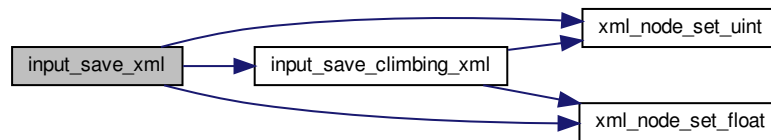
```

```

00339     xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,
00340                         input->threshold);
00341
00342     // Setting the experimental data
00343     for (i = 0; i < input->nexperiments; ++i)
00344     {
00345         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00346         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00347                     (xmlChar *) input->experiment[i].name);
00348         if (input->experiment[i].weight != 1.)
00349             xml_node_set_float (child, (const xmlChar *)
LABEL_WEIGHT,
00350                                 input->experiment[i].weight);
00351         for (j = 0; j < input->experiment->ninputs; ++j)
00352             xmlSetProp (child, (const xmlChar *) stencil[j],
00353                         (xmlChar *) input->experiment[i].stencil[j]);
00354     }
00355
00356     // Setting the variables data
00357     for (i = 0; i < input->nvariables; ++i)
00358     {
00359         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00360         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361                     (xmlChar *) input->variable[i].name);
00362         xml_node_set_float (child, (const xmlChar *)
LABEL_MINIMUM,
00363                             input->variable[i].rangemin);
00364         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00365             xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MINIMUM,
00366                                 input->variable[i].rangeminabs);
00367         xml_node_set_float (child, (const xmlChar *)
LABEL_MAXIMUM,
00368                             input->variable[i].rangemax);
00369         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370             xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MAXIMUM,
00371                                 input->variable[i].rangemaxabs);
00372         if (input->variable[i].precision !=
DEFAULT_PRECISION)
00373             xml_node_set_uint (child, (const xmlChar *)
LABEL_PRECISION,
00374                                 input->variable[i].precision);
00375         if (input->algorithm == ALGORITHM_SWEEP
00376             || input->algorithm == ALGORITHM_ORTHOGONAL)
00377             xml_node_set_uint (child, (const xmlChar *)
LABEL_NSWEEPS,
00378                                 input->variable[i].nsweeps);
00379         else if (input->algorithm == ALGORITHM_GENETIC)
00380             xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00381                                 input->variable[i].nbits);
00382         if (input->nsteps)
00383             xml_node_set_float (child, (const xmlChar *)
LABEL_STEP,
00384                                 input->variable[i].step);
00385     }
00386
00387     // Saving the error norm
00388     switch (input->norm)
00389     {
00390     case ERROR_NORM_MAXIMUM:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                     (const xmlChar *) LABEL_MAXIMUM);
00393         break;
00394     case ERROR_NORM_P:
00395         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                     (const xmlChar *) LABEL_P);
00397         xml_node_set_float (node, (const xmlChar *) LABEL_P,
input->p);
00398         break;
00399     case ERROR_NORM_TAXICAB:
00400         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                     (const xmlChar *) LABEL_TAXICAB);
00402     }
00403
00404     #if DEBUG_INTERFACE
00405     fprintf (stderr, "input_save: end\n");
00406     #endif
00407 }

```

Here is the call graph for this function:



4.11.2.6 options_new()

```
void options_new ( )
```

Function to open the options dialog.

Definition at line 632 of file [interface.c](#).

```

00633 {
00634 #if DEBUG_INTERFACE
00635     fprintf (stderr, "options_new: start\n");
00636 #endif
00637     options->label_seed = (GtkLabel *)
00638         gtk_label_new (_("Pseudo-random numbers generator seed"));
00639     options->spin_seed = (GtkSpinButton *)
00640         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00641     gtk_widget_set_tooltip_text
00642         (GTK_WIDGET (options->spin_seed),
00643          _("Seed to init the pseudo-random numbers generator"));
00644     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00645     options->label_threads = (GtkLabel *)
00646         gtk_label_new (_("Threads number for the stochastic algorithm"));
00647     options->spin_threads
00648         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00649     gtk_widget_set_tooltip_text
00650         (GTK_WIDGET (options->spin_threads),
00651          _("Number of threads to perform the calibration/optimization for "
00652            "the stochastic algorithm"));
00653     gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00654     options->label_climbing = (GtkLabel *)
00655         gtk_label_new (_("Threads number for the hill climbing method"));
00656     options->spin_climbing =
00657         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00658     gtk_widget_set_tooltip_text
00659         (GTK_WIDGET (options->spin_climbing),
00660          _("Number of threads to perform the calibration/optimization for the "
00661            "hill climbing method"));
00662     gtk_spin_button_set_value (options->spin_climbing,
00663                               (gdouble) nthreads_climbing);
00664     options->grid = (GtkGrid *) gtk_grid_new ();
00665     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_seed), 0, 0, 1, 1);
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_seed), 1, 0, 1, 1);
00667     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_threads),
00668                     0, 1, 1, 1);
00669     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_threads),
00670                     1, 1, 1, 1);
00671     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_climbing), 0, 2, 1,
00672                     1);
00673     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_climbing), 1, 2, 1,

```



```

00674         1);
00675     gtk_widget_show_all (GTK_WIDGET (options->grid));
00676     options->dialog = (GtkDialog *)
00677         gtk_dialog_new_with_buttons (_("Options"),
00678                                     window->window,
00679                                     GTK_DIALOG_MODAL,
00680                                     _("_OK"), GTK_RESPONSE_OK,
00681                                     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00682     gtk_container_add
00683         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00684          GTK_WIDGET (options->grid));
00685     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00686     {
00687         input->seed
00688             = (unsigned long int) gtk_spin_button_get_value (options->
spin_seed);
00689         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00690         nthreads_climbing
00691             = gtk_spin_button_get_value_as_int (options->spin_climbing);
00692     }
00693     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00694 #if DEBUG_INTERFACE
00695     fprintf (stderr, "options_new: end\n");
00696 #endif
00697 }

```

4.11.2.7 running_new()

```
void running_new ( )
```

Function to open the running dialog.

Definition at line 703 of file [interface.c](#).

```

00704 {
00705 #if DEBUG_INTERFACE
00706     fprintf (stderr, "running_new: start\n");
00707 #endif
00708     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00709     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00710     running->grid = (GtkGrid *) gtk_grid_new ();
00711     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00712     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00713     running->dialog = (GtkDialog *)
00714         gtk_dialog_new_with_buttons (_("Calculating"),
00715                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00716     gtk_container_add (GTK_CONTAINER
00717         (gtk_dialog_get_content_area (running->dialog)),
00718          GTK_WIDGET (running->grid));
00719     gtk_spinner_start (running->spinner);
00720     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721 #if DEBUG_INTERFACE
00722     fprintf (stderr, "running_new: end\n");
00723 #endif
00724 }

```

4.11.2.8 window_about()

```
void window_about ( )
```

Function to show an about dialog.

Definition at line 1057 of file [interface.c](#).

```

01058 {
01059     static const gchar *authors[] = {
01060         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01061         "Borja Latorre Garcés <borja.latorre@csic.es>",
01062         NULL
01063     };
01064     #if DEBUG_INTERFACE
01065     fprintf (stderr, "window_about: start\n");
01066     #endif
01067     gtk_show_about_dialog
01068     (window->window,
01069      "program_name", "MPCOTool",
01070      "comments",
01071      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01072       "A software to perform calibrations or optimizations of empirical "
01073       "parameters"),
01074      "authors", authors,
01075      "translator-credits",
01076      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01077       "(english, french and spanish)\n"
01078       "Uğur Çayoğlu (german)",
01079      "version", "4.0.1",
01080      "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
01081      "logo", window->logo,
01082      "website", "https://github.com/jburguete/mpcotool",
01083      "license-type", GTK_LICENSE_BSD, NULL);
01084     #if DEBUG_INTERFACE
01085     fprintf (stderr, "window_about: end\n");
01086     #endif
01087 }

```

4.11.2.9 window_add_experiment()

```
void window_add_experiment ( )
```

Function to add an experiment in the main window.

Definition at line 1392 of file [interface.c](#).

```

01393 {
01394     unsigned int i, j;
01395     #if DEBUG_INTERFACE
01396     fprintf (stderr, "window_add_experiment: start\n");
01397     #endif
01398     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01399     g_signal_handler_block (window->combo_experiment, window->
01400     id_experiment);
01401     gtk_combo_box_text_insert_text
01402     (window->combo_experiment, i, input->experiment[i].
01403     name);
01404     g_signal_handler_unblock (window->combo_experiment,
01405     window->id_experiment);
01406     input->experiment = (Experiment *) g_realloc
01407     (input->experiment, (input->nexperiments + 1) * sizeof (
01408     Experiment));
01409     for (j = input->nexperiments - 1; j > i; --j)
01410     memcpy (input->experiment + j + 1, input->experiment + j,
01411     sizeof (Experiment));
01412     input->experiment[j + 1].weight = input->experiment[j].
01413     weight;
01414     input->experiment[j + 1].ninputs = input->
01415     experiment[j].ninputs;
01416     if (input->type == INPUT_TYPE_XML)
01417     {
01418         input->experiment[j + 1].name
01419         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
01420         name);
01421         for (j = 0; j < input->experiment->ninputs; ++j)
01422             input->experiment[i + 1].stencil[j]
01423             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
01424             stencil[j]);
01425     }
01426     else
01427     {
01428         input->experiment[j + 1].name = g_strdup (input->

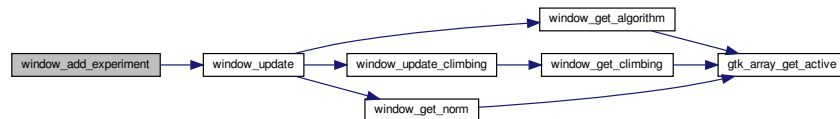
```

```

    experiment[j].name);
01421     for (j = 0; j < input->experiment->ninputs; ++j)
01422         input->experiment[i + 1].stencil[j]
01423         = g_strdup (input->experiment[i].stencil[j]);
01424     }
01425     ++input->nexperiments;
01426     for (j = 0; j < input->experiment->ninputs; ++j)
01427         g_signal_handler_block (window->button_template[j],
window->id_input[j]);
01428     g_signal_handler_block
01429     (window->button_experiment, window->
id_experiment_name);
01430     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01431     g_signal_handler_unblock
01432     (window->button_experiment, window->
id_experiment_name);
01433     for (j = 0; j < input->experiment->ninputs; ++j)
01434         g_signal_handler_unblock (window->button_template[j],
window->id_input[j]);
01435     window_update ();
01436     #if DEBUG_INTERFACE
01437     fprintf (stderr, "window_add_experiment: end\n");
01438     #endif
01439 }

```

Here is the call graph for this function:



4.11.2.10 window_add_variable()

```
void window_add_variable ( )
```

Function to add a variable in the main window.

Definition at line 1655 of file [interface.c](#).

```

01656 {
01657     unsigned int i, j;
01658     #if DEBUG_INTERFACE
01659     fprintf (stderr, "window_add_variable: start\n");
01660     #endif
01661     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01662     g_signal_handler_block (window->combo_variable, window->
id_variable);
01663     gtk_combo_box_text_insert_text (window->combo_variable, i,
01664                                     input->variable[i].name);
01665     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01666     input->variable = (Variable *) g_realloc
01667     (input->variable, (input->nvariables + 1) * sizeof (
Variable));
01668     for (j = input->nvariables - 1; j > i; --j)
01669         memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01670     memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01671     if (input->type == INPUT_TYPE_XML)
01672         input->variable[j + 1].name
01673         = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01674     else
01675         input->variable[j + 1].name = g_strdup (input->

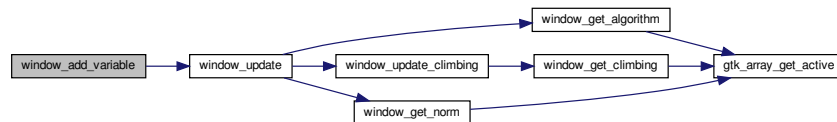
```

```

        variable[j].name);
01676     ++input->nvariables;
01677     g_signal_handler_block (window->entry_variable, window->
        id_variable_label);
01678     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01679     g_signal_handler_unblock (window->entry_variable, window->
        id_variable_label);
01680     window_update ();
01681     #if DEBUG_INTERFACE
01682     fprintf (stderr, "window_add_variable: end\n");
01683     #endif
01684 }

```

Here is the call graph for this function:



4.11.2.11 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 732 of file [interface.c](#).

```

00733 {
00734     unsigned int i;
00735     #if DEBUG_INTERFACE
00736     fprintf (stderr, "window_get_algorithm: start\n");
00737     #endif
00738     i = gtk_array_get_active (window->button_algorithm,
        NALGORITHMS);
00739     #if DEBUG_INTERFACE
00740     fprintf (stderr, "window_get_algorithm: %u\n", i);
00741     fprintf (stderr, "window_get_algorithm: end\n");
00742     #endif
00743     return i;
00744 }

```

Here is the call graph for this function:



4.11.2.12 window_get_climbing()

```
unsigned int window_get_climbing ( )
```

Function to get the hill climbing method number.

Returns

Hill climbing method number.

Definition at line 752 of file [interface.c](#).

```
00753 {  
00754     unsigned int i;  
00755     #if DEBUG_INTERFACE  
00756     fprintf (stderr, "window_get_climbing: start\n");  
00757     #endif  
00758     i = gtk_array_get_active (window->button_climbing,  
00759                             NCLIMBINGS);  
00759     #if DEBUG_INTERFACE  
00760     fprintf (stderr, "window_get_climbing: %u\n", i);  
00761     fprintf (stderr, "window_get_climbing: end\n");  
00762     #endif  
00763     return i;  
00764 }
```

Here is the call graph for this function:



4.11.2.13 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

Returns

Norm method number.

Definition at line 772 of file [interface.c](#).

```

00773 {
00774     unsigned int i;
00775     #if DEBUG_INTERFACE
00776     fprintf (stderr, "window_get_norm: start\n");
00777     #endif
00778     i = gtk_array_get_active (window->button_norm,
NNORMS);
00779     #if DEBUG_INTERFACE
00780     fprintf (stderr, "window_get_norm: %u\n", i);
00781     fprintf (stderr, "window_get_norm: end\n");
00782     #endif
00783     return i;
00784 }

```

Here is the call graph for this function:



4.11.2.14 window_help()

```
void window_help ( )
```

Function to show a help dialog.

Definition at line 1029 of file [interface.c](#).

```

01030 {
01031     char *buffer, *buffer2;
01032     #if DEBUG_INTERFACE
01033     fprintf (stderr, "window_help: start\n");
01034     #endif
01035     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01036                               _("user-manual.pdf"), NULL);
01037     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01038     g_free (buffer2);
01039     #if GTK_MINOR_VERSION >= 22
01040     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01041     #else
01042     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01043     #endif
01044     #if DEBUG_INTERFACE
01045     fprintf (stderr, "window_help: uri=%s\n", buffer);
01046     #endif
01047     g_free (buffer);
01048     #if DEBUG_INTERFACE
01049     fprintf (stderr, "window_help: end\n");
01050     #endif
01051 }

```

4.11.2.15 window_inputs_experiment()

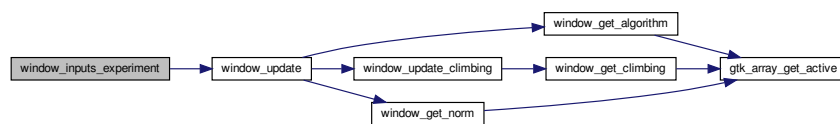
```
void window_inputs_experiment ( )
```

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

Definition at line 1492 of file [interface.c](#).

```
01493 {
01494     unsigned int j;
01495     #if DEBUG_INTERFACE
01496     fprintf (stderr, "window_inputs_experiment: start\n");
01497     #endif
01498     j = input->experiment->ninputs - 1;
01499     if (j
01500         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01501                                           (window->check_template[j]))
01502         --input->experiment->ninputs;
01503     if (input->experiment->ninputs < MAX_NINPUTS
01504         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01505                                           (window->check_template[j]))
01506         ++input->experiment->ninputs;
01507     window_update ();
01508     #if DEBUG_INTERFACE
01509     fprintf (stderr, "window_inputs_experiment: end\n");
01510     #endif
01511 }
```

Here is the call graph for this function:



4.11.2.16 window_label_variable()

```
void window_label_variable ( )
```

Function to set the variable label in the main window.

Definition at line 1690 of file [interface.c](#).

```
01691 {
01692     unsigned int i;
01693     const char *buffer;
01694     #if DEBUG_INTERFACE
01695     fprintf (stderr, "window_label_variable: start\n");
01696     #endif
01697     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01698     buffer = gtk_entry_get_text (window->entry_variable);
01699     g_signal_handler_block (window->combo_variable, window->
01700                           id_variable);
01701     gtk_combo_box_text_remove (window->combo_variable, i);
01702     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01703     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01704     g_signal_handler_unblock (window->combo_variable, window->
01705                             id_variable);
01706     #if DEBUG_INTERFACE
01707     fprintf (stderr, "window_label_variable: end\n");
01708     #endif
01709 }
```

4.11.2.17 window_name_experiment()

```
void window_name_experiment ( )
```

Function to set the experiment name in the main window.

Definition at line 1445 of file [interface.c](#).

```
01446 {
01447     unsigned int i;
01448     char *buffer;
01449     GFile *file1, *file2;
01450     #if DEBUG_INTERFACE
01451     fprintf (stderr, "window_name_experiment: start\n");
01452     #endif
01453     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01454     file1
01455         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
button_experiment));
01456     file2 = g_file_new_for_path (input->directory);
01457     buffer = g_file_get_relative_path (file2, file1);
01458     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01459     gtk_combo_box_text_remove (window->combo_experiment, i);
01460     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01461     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01462     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01463     g_free (buffer);
01464     g_object_unref (file2);
01465     g_object_unref (file1);
01466     #if DEBUG_INTERFACE
01467     fprintf (stderr, "window_name_experiment: end\n");
01468     #endif
01469 }
```

4.11.2.18 window_new()

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

Function to open the main window.

Parameters

<i>application</i>	GtkApplication struct.
--------------------	------------------------

Definition at line 2065 of file [interface.c](#).

```
02066 {
02067     unsigned int i;
02068     char *buffer, *buffer2, buffer3[64];
02069     char *label_algorithm[NALGORITHMS] = {
02070         "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02071     };
02072     char *tip_algorithm[NALGORITHMS] = {
02073         _("Monte-Carlo brute force algorithm"),
02074         _("Sweep brute force algorithm"),
02075         _("Genetic algorithm"),
02076         _("Orthogonal sampling brute force algorithm"),
02077     };
02078     char *label_climbing[NCLIMBINGS] = {
02079         _("_Coordinates climbing"), _("_Random climbing")
02080     };
02081 }
```



```

02081 char *tip_climbing[NCLIMBINGS] = {
02082     _("Coordinates climbing estimate method"),
02083     _("Random climbing estimate method")
02084 };
02085 char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02086 char *tip_norm[NNORMS] = {
02087     _("Euclidean error norm (L2)"),
02088     _("Maximum error norm (L)"),
02089     _("P error norm (Lp)"),
02090     _("Taxicab error norm (L1)")
02091 };
02092
02093 #if DEBUG_INTERFACE
02094 fprintf (stderr, "window_new: start\n");
02095 #endif
02096
02097 // Creating the window
02098 window->window = main_window
02099     = (GtkWindow *) gtk_application_window_new (application);
02100
02101 // Finish when closing the window
02102 g_signal_connect_swapped (window->window, "delete-event",
02103     G_CALLBACK (g_application_quit),
02104     G_APPLICATION (application));
02105
02106 // Setting the window title
02107 gtk_window_set_title (window->window, "MPCOTool");
02108
02109 // Creating the open button
02110 window->button_open = (GtkToolButton *) gtk_tool_button_new
02111     (gtk_image_new_from_icon_name ("document-open",
02112     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02113 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02114
02115 // Creating the save button
02116 window->button_save = (GtkToolButton *) gtk_tool_button_new
02117     (gtk_image_new_from_icon_name ("document-save",
02118     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02119 g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
02120     NULL);
02121
02122 // Creating the run button
02123 window->button_run = (GtkToolButton *) gtk_tool_button_new
02124     (gtk_image_new_from_icon_name ("system-run",
02125     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02126 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02127
02128 // Creating the options button
02129 window->button_options = (GtkToolButton *) gtk_tool_button_new
02130     (gtk_image_new_from_icon_name ("preferences-system",
02131     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02132 g_signal_connect (window->button_options, "clicked",
options_new, NULL);
02133
02134 // Creating the help button
02135 window->button_help = (GtkToolButton *) gtk_tool_button_new
02136     (gtk_image_new_from_icon_name ("help-browser",
02137     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02138 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02139
02140 // Creating the about button
02141 window->button_about = (GtkToolButton *) gtk_tool_button_new
02142     (gtk_image_new_from_icon_name ("help-about",
02143     GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02144 g_signal_connect (window->button_about, "clicked",
window_about, NULL);
02145
02146 // Creating the exit button
02147 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02148     (gtk_image_new_from_icon_name ("application-exit",
02149     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02150 g_signal_connect_swapped (window->button_exit, "clicked",
02151     G_CALLBACK (g_application_quit),
02152     G_APPLICATION (application));
02153
02154 // Creating the buttons bar
02155 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02156 gtk_toolbar_insert
02157     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_open), 0);
02158 gtk_toolbar_insert
02159     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_save), 1);
02160 gtk_toolbar_insert
02161     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_run), 2);

```

```

02162 gtk_toolbar_insert
02163 (window->bar_buttons, GTK_TOOL_ITEM (window->
button_options), 3);
02164 gtk_toolbar_insert
02165 (window->bar_buttons, GTK_TOOL_ITEM (window->
button_help), 4);
02166 gtk_toolbar_insert
02167 (window->bar_buttons, GTK_TOOL_ITEM (window->
button_about), 5);
02168 gtk_toolbar_insert
02169 (window->bar_buttons, GTK_TOOL_ITEM (window->
button_exit), 6);
02170 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02171
02172 // Creating the simulator program label and entry
02173 window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02174 window->button_simulator = (GtkFileChooserButton *)
02175 gtk_file_chooser_button_new (_("Simulator program"),
02176 GTK_FILE_CHOOSER_ACTION_OPEN);
02177 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02178 _("Simulator program executable file"));
02179 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02180
02181 // Creating the evaluator program label and entry
02182 window->check_evaluator = (GtkCheckButton *)
02183 gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02184 g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
02185 window->button_evaluator = (GtkFileChooserButton *)
02186 gtk_file_chooser_button_new (_("Evaluator program"),
02187 GTK_FILE_CHOOSER_ACTION_OPEN);
02188 gtk_widget_set_tooltip_text
02189 (GTK_WIDGET (window->button_evaluator),
02190 _("Optional evaluator program executable file"));
02191
02192 // Creating the results files labels and entries
02193 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02194 window->entry_result = (GtkEntry *) gtk_entry_new ();
02195 gtk_widget_set_tooltip_text
02196 (GTK_WIDGET (window->entry_result), _("Best results file"));
02197 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02198 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02199 gtk_widget_set_tooltip_text
02200 (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02201
02202 // Creating the files grid and attaching widgets
02203 window->grid_files = (GtkGrid *) gtk_grid_new ();
02204 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02205 0, 0, 1, 1);
02206 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02207 1, 0, 1, 1);
02208 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02209 0, 1, 1, 1);
02210 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02211 1, 1, 1, 1);
02212 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02213 0, 2, 1, 1);
02214 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02215 1, 2, 1, 1);
02216 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02217 0, 3, 1, 1);
02218 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02219 1, 3, 1, 1);
02220
02221 // Creating the algorithm properties
02222 window->label_simulations = (GtkLabel *) gtk_label_new
02223 (_("Simulations number"));
02224 window->spin_simulations
02225 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02226 gtk_widget_set_tooltip_text
02227 (GTK_WIDGET (window->spin_simulations),
02228 _("Number of simulations to perform for each iteration"));
02229 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02230 window->label_iterations = (GtkLabel *)
02231 gtk_label_new (_("Iterations number"));
02232 window->spin_iterations
02233 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02234 gtk_widget_set_tooltip_text
02235 (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));

```

```

02236 g_signal_connect
02237 (window->spin_iterations, "value-changed",
window_update, NULL);
02238 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02239 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02240 window->spin_tolerance =
02241 (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02242 gtk_widget_set_tooltip_text
02243 (GTK_WIDGET (window->spin_tolerance),
_ ("Tolerance to set the variable interval on the next iteration"));
02244 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02245 window->spin_bests
02246 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02247 gtk_widget_set_tooltip_text
02248 (GTK_WIDGET (window->spin_bests),
_ ("Number of best simulations used to set the variable interval "
"on the next iteration"));
02250 window->label_population
02251 = (GtkLabel *) gtk_label_new (_("Population number"));
02252 window->spin_population
02253 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02254 gtk_widget_set_tooltip_text
02255 (GTK_WIDGET (window->spin_population),
_ ("Number of population for the genetic algorithm"));
02256 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02257 window->label_generations
02258 = (GtkLabel *) gtk_label_new (_("Generations number"));
02259 window->spin_generations
02260 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02261 gtk_widget_set_tooltip_text
02262 (GTK_WIDGET (window->spin_generations),
_ ("Number of generations for the genetic algorithm"));
02263 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02264 window->spin_mutation
02265 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02266 gtk_widget_set_tooltip_text
02267 (GTK_WIDGET (window->spin_mutation),
_ ("Ratio of mutation for the genetic algorithm"));
02268 window->label_reproduction
02269 = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02270 window->spin_reproduction
02271 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02272 gtk_widget_set_tooltip_text
02273 (GTK_WIDGET (window->spin_reproduction),
_ ("Ratio of reproduction for the genetic algorithm"));
02274 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02275 window->spin_adaptation
02276 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02277 gtk_widget_set_tooltip_text
02278 (GTK_WIDGET (window->spin_adaptation),
_ ("Ratio of adaptation for the genetic algorithm"));
02279 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02280 window->spin_threshold = (GtkSpinButton *)
02281 gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
precision[DEFAULT_PRECISION]);
02282 gtk_widget_set_tooltip_text
02283 (GTK_WIDGET (window->spin_threshold),
_ ("Threshold in the objective function to finish the simulations"));
02284 window->scrolled_threshold =
02285 (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02286 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
GTK_WIDGET (window->spin_threshold));
02287 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02288 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
// GTK_ALIGN_FILL);
02289
02290 // Creating the hill climbing method properties
02291 window->check_climbing = (GtkCheckButton *)
02292 gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02293 g_signal_connect (window->check_climbing, "clicked",
window_update, NULL);
02294 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02295 window->button_climbing[0] = (GtkRadioButton *)
02296 gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02297 gtk_grid_attach (window->grid_climbing,
GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02298 g_signal_connect (window->button_climbing[0], "clicked",
window_update, NULL);
02299 for (i = 0; ++i < NCLIMBINGS;)
02300 {
02301 window->button_climbing[i] = (GtkRadioButton *)
02302 gtk_radio_button_new_with_mnemonic
02303 (gtk_radio_button_get_group (window->button_climbing[0]),
label_climbing[i]);
02304 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
tip_climbing[i]);
02305 gtk_grid_attach (window->grid_climbing,

```

```

02320         GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02321     g_signal_connect (window->button_climbing[i], "clicked",
window_update,
02322         NULL);
02323 }
02324 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02325 window->spin_steps = (GtkSpinButton *)
02326     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02327 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02328 window->label_estimates
02329     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02330 window->spin_estimates = (GtkSpinButton *)
02331     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02332 window->label_relaxation
02333     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02334 window->spin_relaxation = (GtkSpinButton *)
02335     gtk_spin_button_new_with_range (0., 2., 0.001);
02336 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_steps),
02337     0, NCLIMBINGS, 1, 1);
02338 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_steps),
02339     1, NCLIMBINGS, 1, 1);
02340 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_estimates),
02341     0, NCLIMBINGS + 1, 1, 1);
02342 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_estimates),
02343     1, NCLIMBINGS + 1, 1, 1);
02344 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_relaxation),
02345     0, NCLIMBINGS + 2, 1, 1);
02346 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_relaxation),
02347     1, NCLIMBINGS + 2, 1, 1);
02348
02349 // Creating the array of algorithms
02350 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02351 window->button_algorithm[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02353 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02354     tip_algorithm[0]);
02355 gtk_grid_attach (window->grid_algorithm,
02356     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02357 g_signal_connect (window->button_algorithm[0], "clicked",
02358     window_set_algorithm, NULL);
02359 for (i = 0; ++i < NALGORITHMS;)
02360 {
02361     window->button_algorithm[i] = (GtkRadioButton *)
02362         gtk_radio_button_new_with_mnemonic
02363         (gtk_radio_button_get_group (window->button_algorithm[0]),
02364             label_algorithm[i]);
02365     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02366         tip_algorithm[i]);
02367     gtk_grid_attach (window->grid_algorithm,
02368         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02369     g_signal_connect (window->button_algorithm[i], "clicked",
02370         window_set_algorithm, NULL);
02371 }
02372 gtk_grid_attach (window->grid_algorithm,
02373     GTK_WIDGET (window->label_simulations),
02374     0, NALGORITHMS, 1, 1);
02375 gtk_grid_attach (window->grid_algorithm,
02376     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02377 gtk_grid_attach (window->grid_algorithm,
02378     GTK_WIDGET (window->label_iterations),
02379     0, NALGORITHMS + 1, 1, 1);
02380 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_iterations),
02381     1, NALGORITHMS + 1, 1, 1);
02382 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_tolerance),
02383     0, NALGORITHMS + 2, 1, 1);
02384 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_tolerance),
02385     1, NALGORITHMS + 2, 1, 1);
02386 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_bests),
02387     0, NALGORITHMS + 3, 1, 1);
02388 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_bests),
02389     1, NALGORITHMS + 3, 1, 1);
02390 gtk_grid_attach (window->grid_algorithm,
02391     GTK_WIDGET (window->label_population),
02392     0, NALGORITHMS + 4, 1, 1);
02393 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_population),

```

```

02394         1, NALGORITHMS + 4, 1, 1);
02395     gtk_grid_attach (window->grid_algorithm,
02396         GTK_WIDGET (window->label_generations),
02397         0, NALGORITHMS + 5, 1, 1);
02398     gtk_grid_attach (window->grid_algorithm,
02399         GTK_WIDGET (window->spin_generations),
02400         1, NALGORITHMS + 5, 1, 1);
02401     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02402         window->label_mutation),
02403         0, NALGORITHMS + 6, 1, 1);
02404     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02405         window->spin_mutation),
02406         1, NALGORITHMS + 6, 1, 1);
02407     gtk_grid_attach (window->grid_algorithm,
02408         GTK_WIDGET (window->label_reproduction),
02409         0, NALGORITHMS + 7, 1, 1);
02410     gtk_grid_attach (window->grid_algorithm,
02411         GTK_WIDGET (window->spin_reproduction),
02412         1, NALGORITHMS + 7, 1, 1);
02413     gtk_grid_attach (window->grid_algorithm,
02414         GTK_WIDGET (window->label_adaptation),
02415         0, NALGORITHMS + 8, 1, 1);
02416     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02417         window->spin_adaptation),
02418         1, NALGORITHMS + 8, 1, 1);
02419     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02420         window->check_climbing),
02421         0, NALGORITHMS + 9, 2, 1);
02422     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02423         window->grid_climbing),
02424         0, NALGORITHMS + 10, 2, 1);
02425     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02426         window->label_threshold),
02427         0, NALGORITHMS + 11, 1, 1);
02428     gtk_grid_attach (window->grid_algorithm,
02429         GTK_WIDGET (window->scrolled_threshold),
02430         1, NALGORITHMS + 11, 1, 1);
02431     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02432     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02433         GTK_WIDGET (window->grid_algorithm));
02434
02435     // Creating the variable widgets
02436     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02437     gtk_widget_set_tooltip_text
02438         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02439     window->id_variable = g_signal_connect
02440         (window->combo_variable, "changed", window_set_variable, NULL);
02441     window->button_add_variable = (GtkButton *)
02442         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02443     g_signal_connect (window->button_add_variable, "clicked",
02444         window_add_variable,
02445         NULL);
02446     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02447         _("Add variable"));
02448     window->button_remove_variable = (GtkButton *)
02449         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02450     g_signal_connect (window->button_remove_variable, "clicked",
02451         window_remove_variable, NULL);
02452     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02453         _("Remove variable"));
02454     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02455     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02456     gtk_widget_set_tooltip_text
02457         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02458     gtk_widget_set_expand (GTK_WIDGET (window->entry_variable), TRUE);
02459     window->id_variable_label = g_signal_connect
02460         (window->entry_variable, "changed",
02461         window_label_variable, NULL);
02462     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02463     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02464         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02465     gtk_widget_set_tooltip_text
02466         (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02467     window->scrolled_min
02468         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02469     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02470         GTK_WIDGET (window->spin_min));
02471     g_signal_connect (window->spin_min, "value-changed",
02472         window_rangemin_variable, NULL);
02473     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02474     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02475         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02476     gtk_widget_set_tooltip_text
02477         (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02478     window->scrolled_max
02479         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02480     gtk_container_add (GTK_CONTAINER (window->scrolled_max),

```

```

02473         GTK_WIDGET (window->spin_max));
02474 g_signal_connect (window->spin_max, "value-changed",
02475                  window_rangemax_variable, NULL);
02476 window->check_minabs = (GtkCheckButton *)
02477   gtk_check_button_new_with_mnemonic (_("Absolute minimum"));
02478 g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
02479 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02480   (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02481 gtk_widget_set_tooltip_text
02482   (GTK_WIDGET (window->spin_minabs),
02483    _("Minimum allowed value of the variable"));
02484 window->scrolled_minabs
02485   = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02486 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02487                   GTK_WIDGET (window->spin_minabs));
02488 g_signal_connect (window->spin_minabs, "value-changed",
02489                  window_rangeminabs_variable, NULL);
02490 window->check_maxabs = (GtkCheckButton *)
02491   gtk_check_button_new_with_mnemonic (_("Absolute maximum"));
02492 g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02493 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02494   (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02495 gtk_widget_set_tooltip_text
02496   (GTK_WIDGET (window->spin_maxabs),
02497    _("Maximum allowed value of the variable"));
02498 window->scrolled_maxabs
02499   = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02500 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02501                   GTK_WIDGET (window->spin_maxabs));
02502 g_signal_connect (window->spin_maxabs, "value-changed",
02503                  window_rangemaxabs_variable, NULL);
02504 window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02505 window->spin_precision = (GtkSpinButton *)
02506   gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02507 gtk_widget_set_tooltip_text
02508   (GTK_WIDGET (window->spin_precision),
02509    _("Number of precision floating point digits\n"
02510      "0 is for integer numbers"));
02511 g_signal_connect (window->spin_precision, "value-changed",
02512                  window_precision_variable, NULL);
02513 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02514 window->spin_sweeps =
02515   (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02516 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02517   _("Number of steps sweeping the variable"));
02518 g_signal_connect (window->spin_sweeps, "value-changed",
02519                  window_update_variable, NULL);
02520 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02521 window->spin_bits
02522   = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02523 gtk_widget_set_tooltip_text
02524   (GTK_WIDGET (window->spin_bits),
02525    _("Number of bits to encode the variable"));
02526 g_signal_connect
02527   (window->spin_bits, "value-changed", window_update_variable, NULL);
02528 ;
02529 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02530 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02531   (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02532 gtk_widget_set_tooltip_text
02533   (GTK_WIDGET (window->spin_step),
02534    _("Initial step size for the hill climbing method"));
02535 window->scrolled_step
02536   = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02537 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02538                   GTK_WIDGET (window->spin_step));
02539 g_signal_connect
02540   (window->spin_step, "value-changed", window_step_variable, NULL);
02541 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02542 gtk_grid_attach (window->grid_variable,
02543                 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02544 gtk_grid_attach (window->grid_variable,
02545                 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02546 gtk_grid_attach (window->grid_variable,
02547                 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02548 gtk_grid_attach (window->grid_variable,
02549                 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02550 gtk_grid_attach (window->grid_variable,
02551                 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02552 gtk_grid_attach (window->grid_variable,
02553                 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02554 gtk_grid_attach (window->grid_variable,
02555                 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02556 gtk_grid_attach (window->grid_variable,
02557                 GTK_WIDGET (window->label_max), 0, 3, 1, 1);

```

```

02557 gtk_grid_attach (window->grid_variable,
02558                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02559 gtk_grid_attach (window->grid_variable,
02560                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02561 gtk_grid_attach (window->grid_variable,
02562                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02563 gtk_grid_attach (window->grid_variable,
02564                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02565 gtk_grid_attach (window->grid_variable,
02566                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02567 gtk_grid_attach (window->grid_variable,
02568                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02569 gtk_grid_attach (window->grid_variable,
02570                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02571 gtk_grid_attach (window->grid_variable,
02572                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02573 gtk_grid_attach (window->grid_variable,
02574                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02575 gtk_grid_attach (window->grid_variable,
02576                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02577 gtk_grid_attach (window->grid_variable,
02578                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02579 gtk_grid_attach (window->grid_variable,
02580                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02581 gtk_grid_attach (window->grid_variable,
02582                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02583 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02584 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02585                   GTK_WIDGET (window->grid_variable));
02586
02587 // Creating the experiment widgets
02588 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02589 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02590                             _("Experiment selector"));
02591 window->id_experiment = g_signal_connect
02592 (window->combo_experiment, "changed",
02593  window_set_experiment, NULL);
02594 window->button_add_experiment = (GtkButton *)
02595   gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02596 g_signal_connect
02597 (window->button_add_experiment, "clicked",
02598  window_add_experiment, NULL);
02599 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02600                             _("Add experiment"));
02601 window->button_remove_experiment = (GtkButton *)
02602   gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02603 g_signal_connect (window->button_remove_experiment, "clicked",
02604                  window_remove_experiment, NULL);
02605 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
02606   button_remove_experiment),
02607                             _("Remove experiment"));
02608 window->label_experiment
02609   = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02610 window->button_experiment = (GtkFileChooserButton *)
02611   gtk_file_chooser_button_new (_("Experimental data file"),
02612                               GTK_FILE_CHOOSER_ACTION_OPEN);
02613 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02614                             _("Experimental data file"));
02615 window->id_experiment_name
02616   = g_signal_connect (window->button_experiment, "selection-changed",
02617                      window_name_experiment, NULL);
02618 gtk_widget_set_hexspan (GTK_WIDGET (window->button_experiment), TRUE);
02619 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02620 window->spin_weight
02621   = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02622 gtk_widget_set_tooltip_text
02623 (GTK_WIDGET (window->spin_weight),
02624  _("Weight factor to build the objective function"));
02625 g_signal_connect
02626 (window->spin_weight, "value-changed",
02627  window_weight_experiment, NULL);
02628 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02629 gtk_grid_attach (window->grid_experiment,
02630                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02631 gtk_grid_attach (window->grid_experiment,
02632                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02633 gtk_grid_attach (window->grid_experiment,
02634                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02635
02636 gtk_grid_attach (window->grid_experiment,
02637                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02638 gtk_grid_attach (window->grid_experiment,
02639                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02640 gtk_grid_attach (window->grid_experiment,
02641                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02642 gtk_grid_attach (window->grid_experiment,
02643                 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);

```



```

02639     for (i = 0; i < MAX_NINPUTS; ++i)
02640     {
02641         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02642         window->check_template[i] = (GtkCheckButton *)
02643             gtk_check_button_new_with_label (buffer3);
02644         window->id_template[i]
02645             = g_signal_connect (window->check_template[i], "toggled",
02646                 window_inputs_experiment, NULL);
02647         gtk_grid_attach (window->grid_experiment,
02648             GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02649         window->button_template[i] = (GtkFileChooserButton *)
02650             gtk_file_chooser_button_new (_("Input template"),
02651                 GTK_FILE_CHOOSER_ACTION_OPEN);
02652         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02653             _("Experimental input template file"));
02654         window->id_input[i] =
02655             g_signal_connect_swapped (window->button_template[i],
02656                 "selection-changed",
02657                 (GCallback) window_template_experiment,
02658                 (void *) (size_t) i);
02659         gtk_grid_attach (window->grid_experiment,
02660             GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02661     }
02662     window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02663     gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02664         GTK_WIDGET (window->grid_experiment));
02665
02666     // Creating the error norm widgets
02667     window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02668     window->grid_norm = (GtkGrid *) gtk_grid_new ();
02669     gtk_container_add (GTK_CONTAINER (window->frame_norm),
02670         GTK_WIDGET (window->grid_norm));
02671     window->button_norm[0] = (GtkRadioButton *)
02672         gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02673     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02674         tip_norm[0]);
02675     gtk_grid_attach (window->grid_norm,
02676         GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02677     g_signal_connect (window->button_norm[0], "clicked",
02678         window_update, NULL);
02679     for (i = 0; ++i < NNORMS;)
02680     {
02681         window->button_norm[i] = (GtkRadioButton *)
02682             gtk_radio_button_new_with_mnemonic
02683             (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02684         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02685             tip_norm[i]);
02686         gtk_grid_attach (window->grid_norm,
02687             GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02688         g_signal_connect (window->button_norm[i], "clicked",
02689             window_update, NULL);
02690     }
02691     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02692     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02693         label_p), 1, 1, 1, 1);
02694     window->spin_p = (GtkSpinButton *)
02695         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02696     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02697         _("P parameter for the P error norm"));
02698     window->scrolled_p =
02699         (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02700     gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02701         GTK_WIDGET (window->spin_p));
02702     gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02703     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02704     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02705         scrolled_p),
02706         1, 2, 1, 2);
02707
02708     // Creating the grid and attaching the widgets to the grid
02709     window->grid = (GtkGrid *) gtk_grid_new ();
02710     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02711     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02712     gtk_grid_attach (window->grid,
02713         GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02714     gtk_grid_attach (window->grid,
02715         GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02716     gtk_grid_attach (window->grid,
02717         GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02718     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02719     gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
02720         window->grid));
02721
02722     // Setting the window logo
02723     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02724     gtk_window_set_icon (window->window, window->logo);
02725

```



```

02721 // Showing the window
02722 gtk_widget_show_all (GTK_WIDGET (window->window));
02723
02724 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02725 #if GTK_MINOR_VERSION >= 16
02726 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02727 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02728 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02729 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02730 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02731 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02732 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02733 #endif
02734
02735 // Reading initial example
02736 input_new ();
02737 buffer2 = g_get_current_dir ();
02738 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02739 g_free (buffer2);
02740 window_read (buffer);
02741 g_free (buffer);
02742
02743 #if DEBUG_INTERFACE
02744 fprintf (stderr, "window_new: start\n");
02745 #endif
02746 }

```

4.11.2.19 window_open()

```
void window_open ( )
```

Function to open the input data.

Definition at line 1979 of file [interface.c](#).

```

01980 {
01981     GtkFileChooserDialog *dlg;
01982     GtkFileFilter *filter;
01983     char *buffer, *directory, *name;
01984
01985     #if DEBUG_INTERFACE
01986         fprintf (stderr, "window_open: start\n");
01987     #endif
01988
01989     // Saving a backup of the current input file
01990     directory = g_strdup (input->directory);
01991     name = g_strdup (input->name);
01992
01993     // Opening dialog
01994     dlg = (GtkFileChooserDialog *)
01995         gtk_file_chooser_dialog_new (_("Open input file"),
01996                                     window->window,
01997                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01998                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
01999                                     _("_OK"), GTK_RESPONSE_OK, NULL);
02000
02001     // Adding XML filter
02002     filter = (GtkFileFilter *) gtk_file_filter_new ();
02003     gtk_file_filter_set_name (filter, "XML");
02004     gtk_file_filter_add_pattern (filter, "*.xml");
02005     gtk_file_filter_add_pattern (filter, "*.XML");
02006     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02007
02008     // Adding JSON filter
02009     filter = (GtkFileFilter *) gtk_file_filter_new ();
02010     gtk_file_filter_set_name (filter, "JSON");
02011     gtk_file_filter_add_pattern (filter, "*.json");
02012     gtk_file_filter_add_pattern (filter, "*.JSON");
02013     gtk_file_filter_add_pattern (filter, "*.js");
02014     gtk_file_filter_add_pattern (filter, "*.JS");
02015     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02016
02017     // If OK saving
02018     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02019     {

```

```

02020
02021 // Trying to open the input file
02022 buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02023 if (!window_read (buffer))
02024 {
02025 #if DEBUG_INTERFACE
02026     fprintf (stderr, "window_open: error reading input file\n");
02027 #endif
02028     g_free (buffer);
02029
02030     // Reading backup file on error
02031     buffer = g_build_filename (directory, name, NULL);
02032     input->result = input->variables = NULL;
02033     if (!input_open (buffer))
02034     {
02035
02036         // Closing on backup file reading error
02037 #if DEBUG_INTERFACE
02038     fprintf (stderr, "window_read: error reading backup file\n");
02039 #endif
02040     g_free (buffer);
02041     break;
02042     }
02043     g_free (buffer);
02044 }
02045 else
02046 {
02047     g_free (buffer);
02048     break;
02049 }
02050 }
02051
02052 // Freeing and closing
02053 g_free (name);
02054 g_free (directory);
02055 gtk_widget_destroy (GTK_WIDGET (dlg));
02056 #if DEBUG_INTERFACE
02057 fprintf (stderr, "window_open: end\n");
02058 #endif
02059 }

```

4.11.2.20 window_precision_variable()

```
void window_precision_variable ( )
```

Function to update the variable precision in the main window.

Definition at line 1713 of file [interface.c](#).

```

01714 {
01715     unsigned int i;
01716 #if DEBUG_INTERFACE
01717     fprintf (stderr, "window_precision_variable: start\n");
01718 #endif
01719     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01720     input->variable[i].precision
01721     = (unsigned int) gtk_spin_button_get_value_as_int (window->
01722     spin_precision);
01723     gtk_spin_button_set_digits (window->spin_min, input->
01724     variable[i].precision);
01725     gtk_spin_button_set_digits (window->spin_max, input->
01726     variable[i].precision);
01727     gtk_spin_button_set_digits (window->spin_minabs,
01728     input->variable[i].precision);
01729     gtk_spin_button_set_digits (window->spin_maxabs,
01730     input->variable[i].precision);
01731 #if DEBUG_INTERFACE
01732     fprintf (stderr, "window_precision_variable: end\n");
01733 #endif
01734 }

```

4.11.2.21 window_rangemax_variable()

```
void window_rangemax_variable ( )
```

Function to update the variable rangemax in the main window.

Definition at line 1754 of file [interface.c](#).

```
01755 {
01756     unsigned int i;
01757     #if DEBUG_INTERFACE
01758     fprintf (stderr, "window_rangemax_variable: start\n");
01759     #endif
01760     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01761     input->variable[i].rangemax = gtk_spin_button_get_value (
        window->spin_max);
01762     #if DEBUG_INTERFACE
01763     fprintf (stderr, "window_rangemax_variable: end\n");
01764     #endif
01765 }
```

4.11.2.22 window_rangemaxabs_variable()

```
void window_rangemaxabs_variable ( )
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1789 of file [interface.c](#).

```
01790 {
01791     unsigned int i;
01792     #if DEBUG_INTERFACE
01793     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01794     #endif
01795     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01796     input->variable[i].rangemaxabs
01797     = gtk_spin_button_get_value (window->spin_maxabs);
01798     #if DEBUG_INTERFACE
01799     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01800     #endif
01801 }
```

4.11.2.23 window_rangemin_variable()

```
void window_rangemin_variable ( )
```

Function to update the variable rangemin in the main window.

Definition at line 1737 of file [interface.c](#).

```
01738 {
01739     unsigned int i;
01740     #if DEBUG_INTERFACE
01741     fprintf (stderr, "window_rangemin_variable: start\n");
01742     #endif
01743     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01744     input->variable[i].rangemin = gtk_spin_button_get_value (
        window->spin_min);
01745     #if DEBUG_INTERFACE
01746     fprintf (stderr, "window_rangemin_variable: end\n");
01747     #endif
01748 }
```

4.11.2.24 window_rangeminabs_variable()

```
void window_rangeminabs_variable ( )
```

Function to update the variable rangeminabs in the main window.

Definition at line 1771 of file [interface.c](#).

```
01772 {
01773     unsigned int i;
01774     #if DEBUG_INTERFACE
01775     fprintf (stderr, "window_rangeminabs_variable: start\n");
01776     #endif
01777     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01778     input->variable[i].rangeminabs
01779     = gtk_spin_button_get_value (window->spin_minabs);
01780     #if DEBUG_INTERFACE
01781     fprintf (stderr, "window_rangeminabs_variable: end\n");
01782     #endif
01783 }
```

4.11.2.25 window_read()

```
int window_read (
    char * filename )
```

Function to read the input data of a file.

Returns

1 on succes, 0 on error.

Parameters

<i>filename</i>	File name.
-----------------	------------

Definition at line 1863 of file [interface.c](#).

```
01864 {
01865     unsigned int i;
01866     char *buffer;
01867     #if DEBUG_INTERFACE
01868     fprintf (stderr, "window_read: start\n");
01869     #endif
01870
01871     // Reading new input file
01872     input_free ();
01873     input->result = input->variables = NULL;
01874     if (!input_open (filename))
01875     {
01876     #if DEBUG_INTERFACE
01877         fprintf (stderr, "window_read: end\n");
01878     #endif
01879         return 0;
01880     }
01881
01882     // Setting GTK+ widgets data
01883     gtk_entry_set_text (window->entry_result, input->result);
01884     gtk_entry_set_text (window->entry_variables, input->
    variables);
```

```

01885     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01886     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01887         (window->button_simulator), buffer);
01888     g_free (buffer);
01889     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01890         (size_t) input->evaluator);
01891     if (input->evaluator)
01892     {
01893         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01894         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01895             (window->button_evaluator), buffer);
01896         g_free (buffer);
01897     }
01898     gtk_toggle_button_set_active
01899         (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01900     switch (input->algorithm)
01901     {
01902     case ALGORITHM_MONTE_CARLO:
01903         gtk_spin_button_set_value (window->spin_simulations,
01904             (gdouble) input->nsimulations);
01905         // fallthrough
01906     case ALGORITHM_SWEEP:
01907     case ALGORITHM_ORTHOGONAL:
01908         gtk_spin_button_set_value (window->spin_iterations,
01909             (gdouble) input->niterations);
01910         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01911         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01912         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01913             (window->check_climbing),
input->nsteps);
01914         if (input->nsteps)
01915         {
01916             gtk_toggle_button_set_active
01917                 (GTK_TOGGLE_BUTTON (window->button_climbing[
input->climbing]),
TRUE);
01918             gtk_spin_button_set_value (window->spin_steps,
01919                 (gdouble) input->nsteps);
01920             gtk_spin_button_set_value (window->spin_relaxation,
01921                 (gdouble) input->relaxation);
01922             switch (input->climbing)
01923             {
01924             case CLIMBING_METHOD_RANDOM:
01925                 gtk_spin_button_set_value (window->spin_estimates,
01926                     (gdouble) input->nestimates);
01927             }
01928         }
01929         break;
01930     default:
01931         gtk_spin_button_set_value (window->spin_population,
01932             (gdouble) input->nsimulations);
01933         gtk_spin_button_set_value (window->spin_generations,
01934             (gdouble) input->niterations);
01935         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01936         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
01937         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
01938     }
01939     gtk_toggle_button_set_active
01940         (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01941     gtk_spin_button_set_value (window->spin_p, input->p);
01942     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01943     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01944     g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
01945     gtk_combo_box_text_remove_all (window->combo_experiment);
01946     for (i = 0; i < input->nexperiments; ++i)
01947         gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i].name);
01948     g_signal_handler_unblock
01949         (window->button_experiment, window->
id_experiment_name);
01950     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01951     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01952     g_signal_handler_block (window->combo_variable, window->
id_variable);
01953     g_signal_handler_block (window->entry_variable, window->

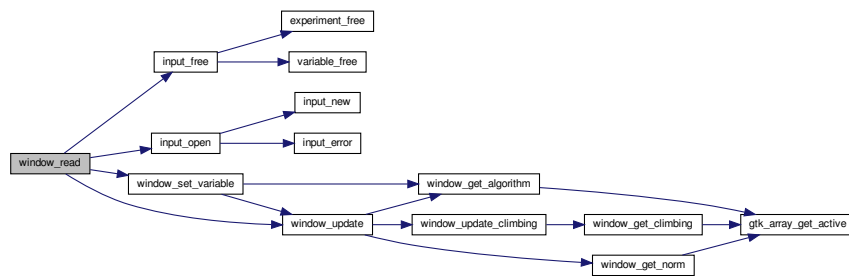
```

```

    id_variable_label);
01959   gtk_combo_box_text_remove_all (window->combo_variable);
01960   for (i = 0; i < input->nvariables; ++i)
01961       gtk_combo_box_text_append_text (window->combo_variable,
01962                                       input->variable[i].name);
01963   g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01964   g_signal_handler_unblock (window->combo_variable, window->
id_variable_label);
01965   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01966   window_set_variable ();
01967   window_update ();
01968
01969   #if DEBUG_INTERFACE
01970       fprintf (stderr, "window_read: end\n");
01971   #endif
01972   return 1;
01973 }

```

Here is the call graph for this function:



4.11.2.26 window_remove_experiment()

```
void window_remove_experiment ( )
```

Function to remove an experiment in the main window.

Definition at line 1355 of file [interface.c](#).

```

01356 {
01357     unsigned int i, j;
01358     #if DEBUG_INTERFACE
01359         fprintf (stderr, "window_remove_experiment: start\n");
01360     #endif
01361     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01362     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01363     gtk_combo_box_text_remove (window->combo_experiment, i);
01364     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01365     experiment_free (input->experiment + i, input->
type);
01366     --input->nexperiments;
01367     for (j = i; j < input->nexperiments; ++j)
01368         memcpy (input->experiment + j, input->experiment + j + 1,
01369               sizeof (Experiment));
01370     j = input->nexperiments - 1;
01371     if (i > j)
01372         i = j;
01373     for (j = 0; j < input->experiment->ninputs; ++j)
01374         g_signal_handler_block (window->button_template[j],
window->id_input[j]);
01375     g_signal_handler_block

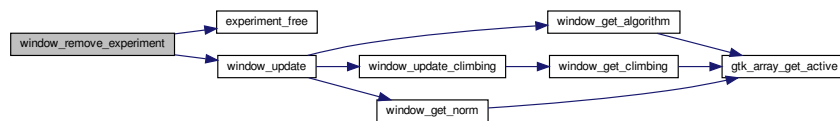
```

```

01376     (window->button_experiment, window->
id_experiment_name);
01377     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01378     g_signal_handler_unblock
01379     (window->button_experiment, window->
id_experiment_name);
01380     for (j = 0; j < input->experiment->ninputs; ++j)
01381         g_signal_handler_unblock (window->button_template[j],
window->id_input[j]);
01382     window_update ();
01383     #if DEBUG_INTERFACE
01384     fprintf (stderr, "window_remove_experiment: end\n");
01385     #endif
01386 }

```

Here is the call graph for this function:



4.11.2.27 window_remove_variable()

```
void window_remove_variable ( )
```

Function to remove a variable in the main window.

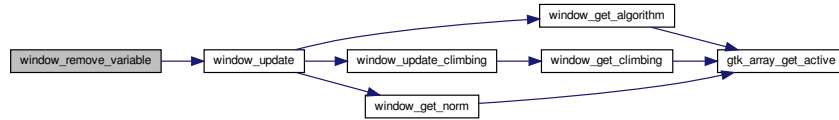
Definition at line 1625 of file [interface.c](#).

```

01626 {
01627     unsigned int i, j;
01628     #if DEBUG_INTERFACE
01629     fprintf (stderr, "window_remove_variable: start\n");
01630     #endif
01631     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01632     g_signal_handler_block (window->combo_variable, window->
id_variable);
01633     gtk_combo_box_text_remove (window->combo_variable, i);
01634     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01635     xmlFree (input->variable[i].name);
01636     --input->nvariables;
01637     for (j = i; j < input->nvariables; ++j)
01638         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01639     j = input->nvariables - 1;
01640     if (i > j)
01641         i = j;
01642     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01643     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01644     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01645     window_update ();
01646     #if DEBUG_INTERFACE
01647     fprintf (stderr, "window_remove_variable: end\n");
01648     #endif
01649 }

```

Here is the call graph for this function:



4.11.2.28 window_run()

```
void window_run ( )
```

Function to run a optimization.

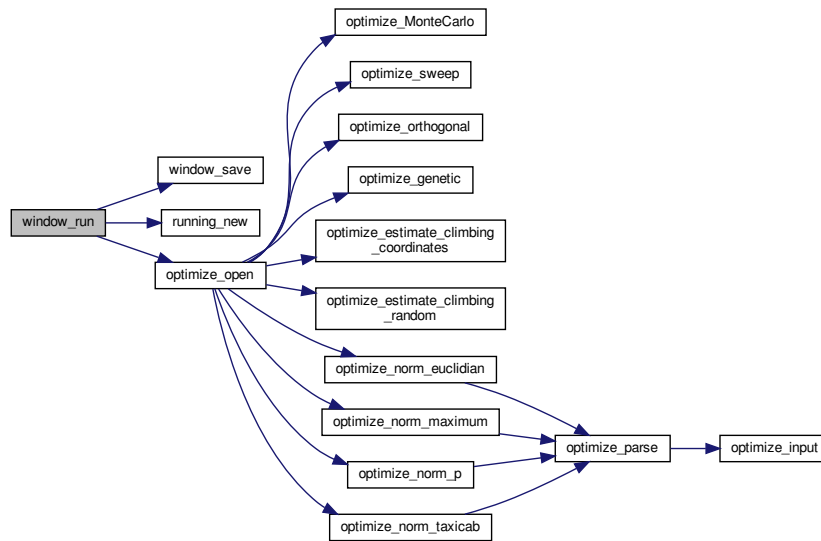
Definition at line 974 of file [interface.c](#).

```

00975 {
00976     unsigned int i;
00977     char *msg, *msg2, buffer[64], buffer2[64];
00978     #if DEBUG_INTERFACE
00979     fprintf (stderr, "window_run: start\n");
00980     #endif
00981     if (!window_save ())
00982     {
00983         #if DEBUG_INTERFACE
00984         fprintf (stderr, "window_run: end\n");
00985         #endif
00986         return;
00987     }
00988     running_new ();
00989     while (gtk_events_pending ())
00990         gtk_main_iteration ();
00991     optimize_open ();
00992     #if DEBUG_INTERFACE
00993     fprintf (stderr, "window_run: closing running dialog\n");
00994     #endif
00995     gtk_spinner_stop (running->spinner);
00996     gtk_widget_destroy (GTK_WIDGET (running->dialog));
00997     #if DEBUG_INTERFACE
00998     fprintf (stderr, "window_run: displaying results\n");
00999     #endif
01000     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01001     msg2 = g_strdup (buffer);
01002     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01003     {
01004         snprintf (buffer, 64, "%s = %s\n",
01005             input->variable[i].name, format[input->
01006 variable[i].precision]);
01007         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008         msg = g_strconcat (msg2, buffer2, NULL);
01009         g_free (msg2);
01010     }
01010     snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01011         optimize->calculation_time);
01012     msg = g_strconcat (msg2, buffer, NULL);
01013     g_free (msg2);
01014     show_message (_("Best result"), msg, INFO_TYPE);
01015     g_free (msg);
01016     #if DEBUG_INTERFACE
01017     fprintf (stderr, "window_run: freeing memory\n");
01018     #endif
01019     optimize_free ();
01020     #if DEBUG_INTERFACE
01021     fprintf (stderr, "window_run: end\n");
01022     #endif
01023 }

```


Here is the call graph for this function:



4.11.2.29 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 823 of file [interface.c](#).

```

00824 {
00825     GtkFileChooserDialog *dlg;
00826     GtkFileFilter *filter1, *filter2;
00827     char *buffer;
00828
00829     #if DEBUG_INTERFACE
00830     fprintf (stderr, "window_save: start\n");
00831     #endif
00832
00833     // Opening the saving dialog
00834     dlg = (GtkFileChooserDialog *)
00835         gtk_file_chooser_dialog_new (_, "Save file",
00836                                     window->window,
00837                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00838                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00839                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00840     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00841     buffer = g_build_filename (input->directory, input->name, NULL);
00842     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00843     g_free (buffer);
00844
00845     // Adding XML filter
00846     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00847     gtk_file_filter_set_name (filter1, "XML");

```

```

00848 gtk_file_filter_add_pattern (filter1, "*.xml");
00849 gtk_file_filter_add_pattern (filter1, "*.XML");
00850 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00851
00852 // Adding JSON filter
00853 filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00854 gtk_file_filter_set_name (filter2, "JSON");
00855 gtk_file_filter_add_pattern (filter2, ".json");
00856 gtk_file_filter_add_pattern (filter2, "*.JSON");
00857 gtk_file_filter_add_pattern (filter2, "*.js");
00858 gtk_file_filter_add_pattern (filter2, "*.JS");
00859 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861 if (input->type == INPUT_TYPE_XML)
00862     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00863 else
00864     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00865
00866 // If OK response then saving
00867 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00868 {
00869     // Setting input file type
00870     filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00871     buffer = (char *) gtk_file_filter_get_name (filter1);
00872     if (!strcmp (buffer, "XML"))
00873         input->type = INPUT_TYPE_XML;
00874     else
00875         input->type = INPUT_TYPE_JSON;
00876
00877     // Adding properties to the root XML node
00878     input->simulator = gtk_file_chooser_get_filename
00879         (GTK_FILE_CHOOSER (window->button_simulator));
00880     if (gtk_toggle_button_get_active
00881         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00882         input->evaluator = gtk_file_chooser_get_filename
00883             (GTK_FILE_CHOOSER (window->button_evaluator));
00884     else
00885         input->evaluator = NULL;
00886     if (input->type == INPUT_TYPE_XML)
00887     {
00888         input->result
00889             = (char *) xmlStrdup ((const xmlChar *)
00890                                     gtk_entry_get_text (window->entry_result));
00891         input->variables
00892             = (char *) xmlStrdup ((const xmlChar *)
00893                                     gtk_entry_get_text (window->
00894 entry_variables));
00895     }
00896     else
00897     {
00898         input->result = g_strdup (gtk_entry_get_text (window->
00899 entry_result));
00900         input->variables =
00901             g_strdup (gtk_entry_get_text (window->entry_variables));
00902     }
00903
00904     // Setting the algorithm
00905     switch (window_get_algorithm ())
00906     {
00907         case ALGORITHM_MONTE_CARLO:
00908             input->algorithm = ALGORITHM_MONTE_CARLO;
00909             input->nsimulations
00910                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00911             input->niterations
00912                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00913             input->tolerance = gtk_spin_button_get_value (window->
00914 spin_tolerance);
00915             input->nbest = gtk_spin_button_get_value_as_int (window->
00916 spin_bests);
00917             window_save_climbing ();
00918             break;
00919         case ALGORITHM_SWEEP:
00920             input->algorithm = ALGORITHM_SWEEP;
00921             input->niterations
00922                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00923             input->tolerance = gtk_spin_button_get_value (window->
00924 spin_tolerance);
00925             input->nbest = gtk_spin_button_get_value_as_int (window->
00926 spin_bests);
00927             window_save_climbing ();
00928             break;
00929         case ALGORITHM_ORTHOGONAL:
00930             input->algorithm = ALGORITHM_ORTHOGONAL;
00931             input->niterations
00932                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00933             input->tolerance = gtk_spin_button_get_value (window->
00934 spin_tolerance);

```

```

00928     input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00929     window_save_climbing ();
00930     break;
00931     default:
00932         input->algorithm = ALGORITHM_GENETIC;
00933         input->nsimulations
00934             = gtk_spin_button_get_value_as_int (window->spin_population);
00935         input->niterations
00936             = gtk_spin_button_get_value_as_int (window->spin_generations);
00937         input->mutation_ratio
00938             = gtk_spin_button_get_value (window->spin_mutation);
00939         input->reproduction_ratio
00940             = gtk_spin_button_get_value (window->spin_reproduction);
00941         input->adaptation_ratio
00942             = gtk_spin_button_get_value (window->spin_adaptation);
00943         break;
00944     }
00945     input->norm = window_get_norm ();
00946     input->p = gtk_spin_button_get_value (window->spin_p);
00947     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00948
00949     // Saving the XML file
00950     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00951     input_save (buffer);
00952
00953     // Closing and freeing memory
00954     g_free (buffer);
00955     gtk_widget_destroy (GTK_WIDGET (dlg));
00956 #if DEBUG_INTERFACE
00957     fprintf (stderr, "window_save: end\n");
00958 #endif
00959     return 1;
00960 }
00961
00962 // Closing and freeing memory
00963 gtk_widget_destroy (GTK_WIDGET (dlg));
00964 #if DEBUG_INTERFACE
00965     fprintf (stderr, "window_save: end\n");
00966 #endif
00967     return 0;
00968 }

```

4.11.2.30 window_save_climbing()

```
void window_save_climbing ( )
```

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

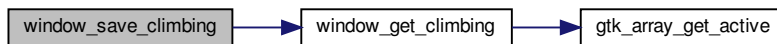
Definition at line 790 of file [interface.c](#).

```

00791 {
00792     #if DEBUG_INTERFACE
00793         fprintf (stderr, "window_save_climbing: start\n");
00794     #endif
00795     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
check_climbing)))
00796     {
00797         input->nsteps = gtk_spin_button_get_value_as_int (window->
spin_steps);
00798         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
00799         switch (window_get_climbing ())
00800         {
00801             case CLIMBING_METHOD_COORDINATES:
00802                 input->climbing = CLIMBING_METHOD_COORDINATES;
00803                 break;
00804             default:
00805                 input->climbing = CLIMBING_METHOD_RANDOM;
00806                 input->nestimates
00807                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00808         }
00809     }
00810     else
00811         input->nsteps = 0;
00812     #if DEBUG_INTERFACE
00813         fprintf (stderr, "window_save_climbing: end\n");
00814     #endif
00815 }

```

Here is the call graph for this function:



4.11.2.31 window_set_algorithm()

```
void window_set_algorithm ( )
```

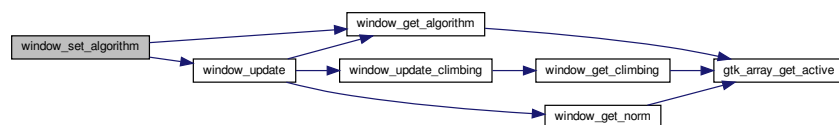
Function to avoid memory errors changing the algorithm.

Definition at line 1281 of file [interface.c](#).

```

01282 {
01283     int i;
01284     #if DEBUG_INTERFACE
01285     fprintf (stderr, "window_set_algorithm: start\n");
01286     #endif
01287     i = window_get_algorithm ();
01288     switch (i)
01289     {
01290     case ALGORITHM_SWEEP:
01291     case ALGORITHM_ORTHOGONAL:
01292         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01293         if (i < 0)
01294             i = 0;
01295         gtk_spin_button_set_value (window->spin_sweeps,
01296                                   (gdouble) input->variable[i].
01297                                   nsweeps);
01298         break;
01299     case ALGORITHM_GENETIC:
01300         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01301         if (i < 0)
01302             i = 0;
01303         gtk_spin_button_set_value (window->spin_bits,
01304                                   (gdouble) input->variable[i].nbits);
01305     }
01306     window_update ();
01307     #if DEBUG_INTERFACE
01308     fprintf (stderr, "window_set_algorithm: end\n");
01309     #endif
01310 }
  
```

Here is the call graph for this function:



4.11.2.32 window_set_experiment()

```
void window_set_experiment ( )
```

Function to set the experiment data in the main window.

Definition at line 1315 of file [interface.c](#).

```
01316 {
01317     unsigned int i, j;
01318     char *buffer1, *buffer2;
01319     #if DEBUG_INTERFACE
01320     fprintf (stderr, "window_set_experiment: start\n");
01321     #endif
01322     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01323     gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
01324     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01325     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01326     g_free (buffer1);
01327     g_signal_handler_block
01328     (window->button_experiment, window->
id_experiment_name);
01329     gtk_file_chooser_set_filename
01330     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01331     g_signal_handler_unblock
01332     (window->button_experiment, window->
id_experiment_name);
01333     g_free (buffer2);
01334     for (j = 0; j < input->experiment->ninputs; ++j)
01335     {
01336         g_signal_handler_block (window->button_template[j],
window->id_input[j]);
01337         buffer2 =
01338         g_build_filename (input->directory, input->experiment[i].
stencil[j],
01339             NULL);
01340         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01341             (window->button_template[j]), buffer2);
01342         g_free (buffer2);
01343         g_signal_handler_unblock
01344         (window->button_template[j], window->id_input[j]);
01345     }
01346     #if DEBUG_INTERFACE
01347     fprintf (stderr, "window_set_experiment: end\n");
01348     #endif
01349 }
```

4.11.2.33 window_set_variable()

```
void window_set_variable ( )
```

Function to set the variable data in the main window.

Definition at line 1548 of file [interface.c](#).

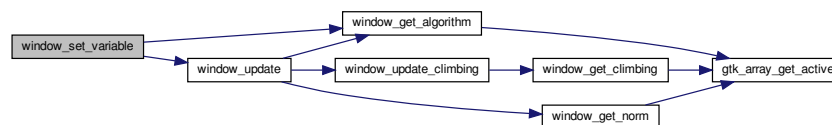
```
01549 {
01550     unsigned int i;
01551     #if DEBUG_INTERFACE
01552     fprintf (stderr, "window_set_variable: start\n");
01553     #endif
01554     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01555     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01556     gtk_entry_set_text (window->entry_variable, input->
variable[i].name);
01557     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01558     gtk_spin_button_set_value (window->spin_min, input->
variable[i].rangemin);
```

```

01559  gtk_spin_button_set_value (window->spin_max, input->
variable[i].rangemax);
01560  if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01561  {
01562      gtk_spin_button_set_value (window->spin_minabs,
01563                               input->variable[i].rangeminabs);
01564      gtk_toggle_button_set_active
01565      (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01566  }
01567  else
01568  {
01569      gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01570      gtk_toggle_button_set_active
01571      (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01572  }
01573  if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01574  {
01575      gtk_spin_button_set_value (window->spin_maxabs,
01576                               input->variable[i].rangemaxabs);
01577      gtk_toggle_button_set_active
01578      (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01579  }
01580  else
01581  {
01582      gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01583      gtk_toggle_button_set_active
01584      (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01585  }
01586  gtk_spin_button_set_value (window->spin_precision,
01587                             input->variable[i].precision);
01588  gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
01589  if (input->nsteps)
01590      gtk_spin_button_set_value (window->spin_step, input->
variable[i].step);
01591  #if DEBUG_INTERFACE
01592      fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01593              input->variable[i].precision);
01594  #endif
01595  switch (window_get_algorithm ())
01596  {
01597      case ALGORITHM_SWEEP:
01598      case ALGORITHM_ORTHOGONAL:
01599          gtk_spin_button_set_value (window->spin_sweeps,
01600                                   (gdouble) input->variable[i].
nsweps);
01601  #if DEBUG_INTERFACE
01602      fprintf (stderr, "window_set_variable: nsweps[%u]=%u\n", i,
01603              input->variable[i].nsweps);
01604  #endif
01605      break;
01606      case ALGORITHM_GENETIC:
01607          gtk_spin_button_set_value (window->spin_bits,
01608                                   (gdouble) input->variable[i].nbits);
01609  #if DEBUG_INTERFACE
01610      fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01611              input->variable[i].nbits);
01612  #endif
01613      break;
01614  }
01615  window_update ();
01616  #if DEBUG_INTERFACE
01617      fprintf (stderr, "window_set_variable: end\n");
01618  #endif
01619  }

```

Here is the call graph for this function:



4.11.234 window_step_variable()

```
void window_step_variable ( )
```

Function to update the variable step in the main window.

Definition at line 1807 of file [interface.c](#).

```
01808 {
01809     unsigned int i;
01810     #if DEBUG_INTERFACE
01811     fprintf (stderr, "window_step_variable: start\n");
01812     #endif
01813     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01814     input->variable[i].step = gtk_spin_button_get_value (window->
    spin_step);
01815     #if DEBUG_INTERFACE
01816     fprintf (stderr, "window_step_variable: end\n");
01817     #endif
01818 }
```

4.11.235 window_template_experiment()

```
void window_template_experiment (
    void * data )
```

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

Parameters

<i>data</i>	Callback data (i-th input template).
-------------	--------------------------------------

Definition at line 1517 of file [interface.c](#).

```
01519 {
01520     unsigned int i, j;
01521     char *buffer;
01522     GFile *file1, *file2;
01523     #if DEBUG_INTERFACE
01524     fprintf (stderr, "window_template_experiment: start\n");
01525     #endif
01526     i = (size_t) data;
01527     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01528     file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01529     file2 = g_file_new_for_path (input->directory);
01530     buffer = g_file_get_relative_path (file2, file1);
01531     if (input->type == INPUT_TYPE_XML)
01532         input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01533     else
01534         input->experiment[j].stencil[i] = g_strdup (buffer);
01535     g_free (buffer);
01536     g_object_unref (file2);
01537     g_object_unref (file1);
01538     #if DEBUG_INTERFACE
01539     fprintf (stderr, "window_template_experiment: end\n");
01540     #endif
01541 }
```

4.11.2.36 window_update()

```
void window_update ( )
```

Function to update the main window view.

Definition at line 1124 of file [interface.c](#).

```
01125 {
01126     unsigned int i;
01127     #if DEBUG_INTERFACE
01128     fprintf (stderr, "window_update: start\n");
01129     #endif
01130     gtk_widget_set_sensitive
01131     (GTK_WIDGET (window->button_evaluator),
01132      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01133      (window->check_evaluator)));
01134     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01135     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01136     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01137     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01138     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01139     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01140     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01142     gtk_widget_hide (GTK_WIDGET (window->label_population));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01144     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01146     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01148     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01149     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01150     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01151     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01152     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01153     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01154     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01155     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01156     gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01157     gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01158     gtk_widget_hide (GTK_WIDGET (window->label_step));
01159     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01160     gtk_widget_hide (GTK_WIDGET (window->label_p));
01161     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01162     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01163     switch (window_get_algorithm ())
01164     {
01165     case ALGORITHM_MONTE_CARLO:
01166         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01167         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01168         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01169         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01170         if (i > 1)
01171         {
01172             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01173             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01174             gtk_widget_show (GTK_WIDGET (window->label_bests));
01175             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01176         }
01177         window_update_climbing ();
01178         break;
01179     case ALGORITHM_SWEEP:
01180     case ALGORITHM_ORTHOGONAL:
01181         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01182         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01183         if (i > 1)
01184         {
01185             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01186             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01187             gtk_widget_show (GTK_WIDGET (window->label_bests));
01188             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01189         }
01190         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01191         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01192         gtk_widget_show (GTK_WIDGET (window->check_climbing));
01193         window_update_climbing ();
01194         break;
01195     default:
01196         gtk_widget_show (GTK_WIDGET (window->label_population));
01197         gtk_widget_show (GTK_WIDGET (window->spin_population));
01198         gtk_widget_show (GTK_WIDGET (window->label_generations));
01199     }
```

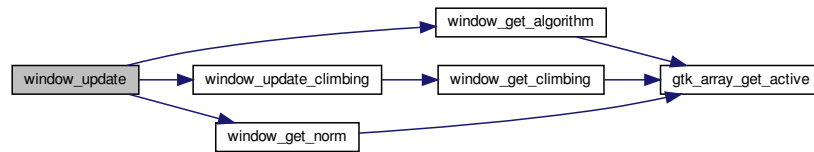


```

01199     gtk_widget_show (GTK_WIDGET (window->spin_generations));
01200     gtk_widget_show (GTK_WIDGET (window->label_mutation));
01201     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01202     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01203     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01204     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01205     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01206     gtk_widget_show (GTK_WIDGET (window->label_bits));
01207     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01208 }
01209 gtk_widget_set_sensitive
01210 (GTK_WIDGET (window->button_remove_experiment),
input->nexperiments > 1);
01211 gtk_widget_set_sensitive
01212 (GTK_WIDGET (window->button_remove_variable),
input->nvariables > 1);
01213 for (i = 0; i < input->experiment->ninputs; ++i)
01214 {
01215     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01216     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01217     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01218     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01219     g_signal_handler_block
01220     (window->check_template[i], window->
id_template[i]);
01221     g_signal_handler_block (window->button_template[i],
window->id_input[i]);
01222     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_template[i]), 1);
01223     g_signal_handler_unblock (window->button_template[i],
01224     window->id_input[i]);
01225     g_signal_handler_unblock (window->check_template[i],
01226     window->id_template[i]);
01227 }
01228 if (i > 0)
01229 {
01230     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01231     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01232     gtk_toggle_button_get_active
01233     GTK_TOGGLE_BUTTON (window->check_template
[i - 1]));
01234 }
01235 if (i < MAX_NINPUTS)
01236 {
01237     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01238     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01239     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01240     gtk_widget_set_sensitive
01241     (GTK_WIDGET (window->button_template[i]),
01242     gtk_toggle_button_get_active
01243     GTK_TOGGLE_BUTTON (window->check_template[i]));
01244     g_signal_handler_block
01245     (window->check_template[i], window->
id_template[i]);
01246     g_signal_handler_block (window->button_template[i],
window->id_input[i]);
01247     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_template[i]), 0);
01248     g_signal_handler_unblock (window->button_template[i],
01249     window->id_input[i]);
01250     g_signal_handler_unblock (window->check_template[i],
01251     window->id_template[i]);
01252 }
01253 while (++i < MAX_NINPUTS)
01254 {
01255     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01256     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01257 }
01258 gtk_widget_set_sensitive
01259 (GTK_WIDGET (window->spin_minabs),
01260     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01261 gtk_widget_set_sensitive
01262 (GTK_WIDGET (window->spin_maxabs),
01263     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01264 if (window_get_norm () == ERROR_NORM_P)
01265 {
01266     gtk_widget_show (GTK_WIDGET (window->label_p));
01267     gtk_widget_show (GTK_WIDGET (window->spin_p));
01268 }
01269 #if DEBUG_INTERFACE
01270 fprintf (stderr, "window_update: end\n");
01271 #endif
01272 }

```

Here is the call graph for this function:



4.11.2.37 window_update_climbing()

```
void window_update_climbing ( )
```

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

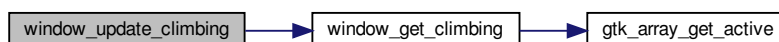
Definition at line 1093 of file [interface.c](#).

```

01094 {
01095     #if DEBUG_INTERFACE
01096         fprintf (stderr, "window_update_climbing: start\n");
01097     #endif
01098     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01099     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
01100         check_climbing)))
01101     {
01102         gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01103         gtk_widget_show (GTK_WIDGET (window->label_step));
01104         gtk_widget_show (GTK_WIDGET (window->spin_step));
01105     }
01106     switch (window_get_climbing ())
01107     {
01108     case CLIMBING_METHOD_COORDINATES:
01109         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01110         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01111         break;
01112     default:
01113         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01114         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01115     }
01116     #if DEBUG_INTERFACE
01117     fprintf (stderr, "window_update_climbing: end\n");
01118     #endif
01119 }

```

Here is the call graph for this function:



4.11.2.38 window_update_variable()

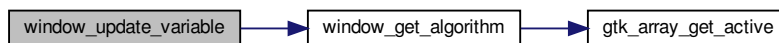
```
void window_update_variable ( )
```

Function to update the variable data in the main window.

Definition at line 1824 of file [interface.c](#).

```
01825 {
01826     int i;
01827     #if DEBUG_INTERFACE
01828     fprintf (stderr, "window_update_variable: start\n");
01829     #endif
01830     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01831     if (i < 0)
01832         i = 0;
01833     switch (window_get_algorithm ())
01834     {
01835     case ALGORITHM_SWEEP:
01836     case ALGORITHM_ORTHOGONAL:
01837         input->variable[i].nsweeps
01838         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01839     #if DEBUG_INTERFACE
01840         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01841                 input->variable[i].nsweeps);
01842     #endif
01843         break;
01844     case ALGORITHM_GENETIC:
01845         input->variable[i].nbits
01846         = gtk_spin_button_get_value_as_int (window->spin_bits);
01847     #if DEBUG_INTERFACE
01848         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01849                 input->variable[i].nbits);
01850     #endif
01851     }
01852     #if DEBUG_INTERFACE
01853     fprintf (stderr, "window_update_variable: end\n");
01854     #endif
01855 }
```

Here is the call graph for this function:



4.11.2.39 window_weight_experiment()

```
void window_weight_experiment ( )
```

Function to update the experiment weight in the main window.

Definition at line 1475 of file [interface.c](#).

```
01476 {
01477     unsigned int i;
01478     #if DEBUG_INTERFACE
01479     fprintf (stderr, "window_weight_experiment: start\n");
01480     #endif
01481     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01482     input->experiment[i].weight = gtk_spin_button_get_value (
01483         window->spin_weight);
01484     #if DEBUG_INTERFACE
01485     fprintf (stderr, "window_weight_experiment: end\n");
01486     #endif
01487 }
```

4.12 interface.c

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #include <gio/gio.h>
00051 #include <gtk/gtk.h>
00052 #include "genetic/genetic.h"
00053 #include "utils.h"
00054 #include "experiment.h"
00055 #include "variable.h"
00056 #include "input.h"
00057 #include "optimize.h"
00058 #include "interface.h"
00059
00060 #define DEBUG_INTERFACE 0
00061
00062 #ifdef G_OS_WIN32
00063 #define INPUT_FILE "test-ga-win.xml"
00064 #else
00065 #define INPUT_FILE "test-ga.xml"
00066 #endif
00067
00068 const char *logo[] = {
00069     "32 32 3 1",
00070     "    c None",
00071     ".    c #0000FF",
00072     "+    c #FF0000",
00073     " ",
00074     " ",
00075     " ",
00076     ".    .    .    .",
00077     ".    .    .    .",
00078     ".    .    .    .",
00079     ".    .    .    .",
00080     ".    .    .    .",
00081     ".    .    .    .",
00082     ".    .    .    .",
00083     ".    .    .    .",
00084     ".    .    .    .",
00085     ".    .    .    .",
00086     ".    .    .    .",
00087     ".    .    .    .",
00088     ".    .    .    .",
00089     ".    .    .    .",
00090     ".    .    .    .",
00091     ".    .    .    .",
00092     ".    .    .    .",
00093     ".    .    .    .",

```

```

00094 "    +++    .    +++    +++    ",
00095 "    +++++    .    .    +++++    ",
00096 "    +++++    .    .    +++++    ",
00097 "    +++++    .    .    +++++    ",
00098 "    +++    .    .    +++    ",
00099 "    .    .    .    .    .    ",
00100 "    .    +++    .    .    .    ",
00101 "    .    +++++    .    .    .    ",
00102 "    .    +++++    .    .    .    ",
00103 "    .    +++++    .    .    .    ",
00104 "    .    +++    .    .    .    ",
00105 "    .    .    .    .    .    ",
00106 "    .    .    .    .    .    ",
00107 "    .    .    .    .    .    ",
00108 "    .    .    .    .    .    ",
00109 "    .    .    .    .    .    ",
00110 "    .    .    .    .    .    ",
00111 "    .    .    .    .    .    ",
00112 "    .    .    .    .    .    ",
00113 "    .    .    .    .    .    ",
00114 "    .    .    .    .    .    ",
00115 };
00116
00117 /*
00118 const char * logo[] = {
00119 "32 32 3 1",
00120 "    c #FFFFFFFFFFFF",
00121 ".    c #00000000FFFF",
00122 "X    c #FFFF00000000",
00123 "    ",
00124 "    ",
00125 "    ",
00126 "    .    .    .    .    .    ",
00127 "    .    .    .    .    .    ",
00128 "    .    .    .    .    .    ",
00129 "    .    .    .    .    .    ",
00130 "    .    .    XXX    .    .    ",
00131 "    .    .    XXXXX    .    .    ",
00132 "    .    .    XXXXX    .    .    ",
00133 "    .    .    XXXXX    .    .    ",
00134 "    XXX    .    XXX    XXX    ",
00135 "    XXXXX    .    .    XXXXX    ",
00136 "    XXXXX    .    .    XXXXX    ",
00137 "    XXXXX    .    .    XXXXX    ",
00138 "    XXX    .    .    XXX    ",
00139 "    .    .    .    .    .    ",
00140 "    .    XXX    .    .    .    ",
00141 "    .    XXXXX    .    .    .    ",
00142 "    .    XXXXX    .    .    .    ",
00143 "    .    XXXXX    .    .    .    ",
00144 "    .    XXX    .    .    .    ",
00145 "    .    .    .    .    .    ",
00146 "    .    .    .    .    .    ",
00147 "    .    .    .    .    .    ",
00148 "    .    .    .    .    .    ",
00149 "    .    .    .    .    .    ",
00150 "    .    .    .    .    .    ",
00151 "    .    .    .    .    .    ",
00152 "    .    .    .    .    .    ",
00153 "    .    .    .    .    .    ",
00154 "    .    .    .    .    .    ";
00155 */
00156
00157 Options options[1];
00159 Running running[1];
00161 Window window[1];
00163
00167 void
00168 input_save_climbing_xml (xmlNode * node)
00169 {
00170     #if DEBUG_INTERFACE
00171         fprintf (stderr, "input_save_climbing_xml: start\n");
00172     #endif
00173     if (input->nsteps)
00174     {
00175         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00176             input->nsteps);
00177         if (input->relaxation != DEFAULT_RELAXATION)
00178             xml_node_set_float (node, (const xmlChar *)
00179                 LABEL_RELAXATION,
00180                 input->relaxation);
00181         switch (input->climbing)
00182         {
00183             case CLIMBING_METHOD_COORDINATES:
00184                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00185                     (const xmlChar *) LABEL_COORDINATES);
00186                 break;

```

```

00185         default:
00186             xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00187                         (const xmlChar *) LABEL_RANDOM);
00188             xml_node_set_uint (node, (const xmlChar *)
00189                               LABEL_NESTIMATES,
00189                               input->nestimates);
00190         }
00191     }
00192     #if DEBUG_INTERFACE
00193     fprintf (stderr, "input_save_climbing_xml: end\n");
00194     #endif
00195 }
00196
00197 void
00200 input_save_climbing_json (JsonNode * node)
00201 {
00202     JsonObject *object;
00203     #if DEBUG_INTERFACE
00204     fprintf (stderr, "input_save_climbing_json: start\n");
00205     #endif
00206     object = json_node_get_object (node);
00207     if (input->nsteps)
00208     {
00209         json_object_set_uint (object, LABEL_NSTEPS,
00210                               input->nsteps);
00211         if (input->relaxation != DEFAULT_RELAXATION)
00212             json_object_set_float (object, LABEL_RELAXATION,
00213                                    input->relaxation);
00214         switch (input->climbing)
00215         {
00216             case CLIMBING_METHOD_COORDINATES:
00217                 json_object_set_string_member (object, LABEL_CLIMBING,
00218                                                LABEL_COORDINATES);
00219                 break;
00220             default:
00221                 json_object_set_string_member (object, LABEL_CLIMBING,
00222                                                LABEL_RANDOM);
00223             json_object_set_uint (object, LABEL_NESTIMATES,
00224                                   input->nestimates);
00225         }
00226     }
00227     #if DEBUG_INTERFACE
00228     fprintf (stderr, "input_save_climbing_json: end\n");
00229     #endif
00230 }
00231
00232 void
00233 input_save_xml (xmlDoc * doc)
00234 {
00235     unsigned int i, j;
00236     char *buffer;
00237     xmlNode *node, *child;
00238     GFile *file, *file2;
00239
00240     #if DEBUG_INTERFACE
00241     fprintf (stderr, "input_save_xml: start\n");
00242     #endif
00243
00244     // Setting root XML node
00245     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00246     xmlDocSetRootElement (doc, node);
00247
00248     // Adding properties to the root XML node
00249     if (xmlStrcmp
00250         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00251         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00252                    (xmlChar *) input->result);
00253     if (xmlStrcmp
00254         ((const xmlChar *) input->variables, (const xmlChar *)
00255          variables_name))
00256         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00257                    (xmlChar *) input->variables);
00258     file = g_file_new_for_path (input->directory);
00259     file2 = g_file_new_for_path (input->simulator);
00260     buffer = g_file_get_relative_path (file, file2);
00261     g_object_unref (file2);
00262     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00263     g_free (buffer);
00264     if (input->evaluator)
00265     {
00266         file2 = g_file_new_for_path (input->evaluator);
00267         buffer = g_file_get_relative_path (file, file2);
00268         g_object_unref (file2);
00269         if (xmlStrlen ((xmlChar *) buffer))
00270             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00271                        (xmlChar *) buffer);
00272         g_free (buffer);
00273     }

```

```

00272     }
00273     if (input->seed != DEFAULT_RANDOM_SEED)
00274         xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);
00275
00276     // Setting the algorithm
00277     buffer = (char *) g_slice_alloc (64);
00278     switch (input->algorithm)
00279     {
00280     case ALGORITHM_MONTE_CARLO:
00281         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00282             (const xmlChar *) LABEL_MONTE_CARLO);
00283         snprintf (buffer, 64, "%u", input->nsimulations);
00284         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00285             (xmlChar *) buffer);
00286         snprintf (buffer, 64, "%u", input->niterations);
00287         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00288             (xmlChar *) buffer);
00289         snprintf (buffer, 64, "%.3lg", input->tolerance);
00290         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00291         snprintf (buffer, 64, "%u", input->nbest);
00292         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00293         input_save_climbing_xml (node);
00294         break;
00295     case ALGORITHM_SWEEP:
00296         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00297             (const xmlChar *) LABEL_SWEEP);
00298         snprintf (buffer, 64, "%u", input->niterations);
00299         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00300             (xmlChar *) buffer);
00301         snprintf (buffer, 64, "%.3lg", input->tolerance);
00302         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00303         snprintf (buffer, 64, "%u", input->nbest);
00304         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00305         input_save_climbing_xml (node);
00306         break;
00307     case ALGORITHM_ORTHOGONAL:
00308         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00309             (const xmlChar *) LABEL_ORTHOGONAL);
00310         snprintf (buffer, 64, "%u", input->niterations);
00311         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00312             (xmlChar *) buffer);
00313         snprintf (buffer, 64, "%.3lg", input->tolerance);
00314         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00315         snprintf (buffer, 64, "%u", input->nbest);
00316         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00317         input_save_climbing_xml (node);
00318         break;
00319     default:
00320         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00321             (const xmlChar *) LABEL_GENETIC);
00322         snprintf (buffer, 64, "%u", input->nsimulations);
00323         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00324             (xmlChar *) buffer);
00325         snprintf (buffer, 64, "%u", input->niterations);
00326         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00327             (xmlChar *) buffer);
00328         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00329         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00332             (xmlChar *) buffer);
00333         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00335         break;
00336     }
00337     g_slice_free1 (64, buffer);
00338     if (input->threshold != 0.)
00339         xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,
input->threshold);
00340
00341     // Setting the experimental data
00342     for (i = 0; i < input->nexperiments; ++i)
00343     {
00344         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00345         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00346             (xmlChar *) input->experiment[i].name);
00347         if (input->experiment[i].weight != 1.)
00348             xml_node_set_float (child, (const xmlChar *)
LABEL_WEIGHT,
input->experiment[i].weight);
00349         for (j = 0; j < input->experiment->ninputs; ++j)
00350             xmlSetProp (child, (const xmlChar *) stencil[j],
00351                 (xmlChar *) input->experiment[i].stencil[j]);
00352     }
00353 }
00354
00355

```

```

00356 // Setting the variables data
00357 for (i = 0; i < input->nvariables; ++i)
00358 {
00359     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00360     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361                 (xmlChar *) input->variable[i].name);
00362     xml_node_set_float (child, (const xmlChar *)
00363 LABEL_MINIMUM,
00364                         input->variable[i].rangemin);
00364     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00365         xml_node_set_float (child, (const xmlChar *)
00366 LABEL_ABSOLUTE_MINIMUM,
00367                             input->variable[i].rangeminabs);
00367     xml_node_set_float (child, (const xmlChar *)
00368 LABEL_MAXIMUM,
00369                         input->variable[i].rangemax);
00369     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
00371 LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00372     if (input->variable[i].precision !=
00373         DEFAULT_PRECISION)
00374         xml_node_set_uint (child, (const xmlChar *)
00375 LABEL_PRECISION,
00376                         input->variable[i].precision);
00376     if (input->algorithm == ALGORITHM_SWEEP
00377         || input->algorithm == ALGORITHM_ORTHOGONAL)
00378         xml_node_set_uint (child, (const xmlChar *)
00379 LABEL_NSWEEPS,
00380                             input->variable[i].nsweeps);
00380     else if (input->algorithm == ALGORITHM_GENETIC)
00381         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00382                             input->variable[i].nbits);
00382     if (input->nsteps)
00383         xml_node_set_float (child, (const xmlChar *)
00384 LABEL_STEP,
00385                             input->variable[i].step);
00385 }
00386
00387 // Saving the error norm
00388 switch (input->norm)
00389 {
00390     case ERROR_NORM_MAXIMUM:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                     (const xmlChar *) LABEL_MAXIMUM);
00393         break;
00394     case ERROR_NORM_P:
00395         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                     (const xmlChar *) LABEL_P);
00397         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00398 input->p);
00399         break;
00399     case ERROR_NORM_TAXICAB:
00400         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                     (const xmlChar *) LABEL_TAXICAB);
00402 }
00403
00404 #if DEBUG_INTERFACE
00405 fprintf (stderr, "input_save: end\n");
00406 #endif
00407 }
00408
00412 void
00413 input_save_json (JsonGenerator * generator)
00414 {
00415     unsigned int i, j;
00416     char *buffer;
00417     JsonNode *node, *child;
00418     JsonObject *object;
00419     JsonArray *array;
00420     GFile *file, *file2;
00421
00422 #if DEBUG_INTERFACE
00423 fprintf (stderr, "input_save_json: start\n");
00424 #endif
00425
00426 // Setting root JSON node
00427 node = json_node_new (JSON_NODE_OBJECT);
00428 object = json_node_get_object (node);
00429 json_generator_set_root (generator, node);
00430
00431 // Adding properties to the root JSON node
00432 if (strcmp (input->result, result_name))
00433     json_object_set_string_member (object, LABEL_RESULT_FILE,
00434 input->result);
00434 if (strcmp (input->variables, variables_name))
00435     json_object_set_string_member (object, LABEL_VARIABLES_FILE,

```



```

00436         input->variables);
00437     file = g_file_new_for_path (input->directory);
00438     file2 = g_file_new_for_path (input->simulator);
00439     buffer = g_file_get_relative_path (file, file2);
00440     g_object_unref (file2);
00441     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00442     g_free (buffer);
00443     if (input->evaluator)
00444     {
00445         file2 = g_file_new_for_path (input->evaluator);
00446         buffer = g_file_get_relative_path (file, file2);
00447         g_object_unref (file2);
00448         if (strlen (buffer))
00449             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00450         g_free (buffer);
00451     }
00452     if (input->seed != DEFAULT_RANDOM_SEED)
00453         json_object_set_uint (object, LABEL_SEED,
input->seed);
00454
00455     // Setting the algorithm
00456     buffer = (char *) g_slice_alloc (64);
00457     switch (input->algorithm)
00458     {
00459     case ALGORITHM_MONTE_CARLO:
00460         json_object_set_string_member (object, LABEL_ALGORITHM,
00461             LABEL_MONTE_CARLO);
00462         snprintf (buffer, 64, "%u", input->nsimulations);
00463         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00464         snprintf (buffer, 64, "%u", input->niterations);
00465         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00466         snprintf (buffer, 64, "%.3lg", input->tolerance);
00467         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00468         snprintf (buffer, 64, "%u", input->nbest);
00469         json_object_set_string_member (object, LABEL_NBEST, buffer);
00470         input_save_climbing_json (node);
00471         break;
00472     case ALGORITHM_SWEEP:
00473         json_object_set_string_member (object, LABEL_ALGORITHM,
00474             LABEL_SWEEP);
00475         snprintf (buffer, 64, "%u", input->niterations);
00476         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00477         snprintf (buffer, 64, "%.3lg", input->tolerance);
00478         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00479         snprintf (buffer, 64, "%u", input->nbest);
00480         json_object_set_string_member (object, LABEL_NBEST, buffer);
00481         input_save_climbing_json (node);
00482         break;
00483     case ALGORITHM_ORTHOGONAL:
00484         json_object_set_string_member (object, LABEL_ALGORITHM,
00485             LABEL_ORTHOGONAL);
00486         snprintf (buffer, 64, "%u", input->niterations);
00487         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00488         snprintf (buffer, 64, "%.3lg", input->tolerance);
00489         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00490         snprintf (buffer, 64, "%u", input->nbest);
00491         json_object_set_string_member (object, LABEL_NBEST, buffer);
00492         input_save_climbing_json (node);
00493         break;
00494     default:
00495         json_object_set_string_member (object, LABEL_ALGORITHM,
00496             LABEL_GENETIC);
00497         snprintf (buffer, 64, "%u", input->nsimulations);
00498         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00499         snprintf (buffer, 64, "%u", input->niterations);
00500         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00501         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00502         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00503         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00504         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00505         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00506         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00507         break;
00508     }
00509     g_slice_free1 (64, buffer);
00510     if (input->threshold != 0.)
00511         json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00512
00513     // Setting the experimental data
00514     array = json_array_new ();
00515     for (i = 0; i < input->nexperiments; ++i)
00516     {
00517         child = json_node_new (JSON_NODE_OBJECT);
00518         object = json_node_get_object (child);
00519         json_object_set_string_member (object, LABEL_NAME,
input->experiment[i].name);

```

```

00518         if (input->experiment[i].weight != 1.)
00519             json_object_set_float (object, LABEL_WEIGHT,
00520                                     input->experiment[i].weight);
00521         for (j = 0; j < input->experiment->ninputs; ++j)
00522             json_object_set_string_member (object, stencil[j],
00523                                             input->experiment[i].
stencil[j]);
00524         json_array_add_element (array, child);
00525     }
00526     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00527
00528     // Setting the variables data
00529     array = json_array_new ();
00530     for (i = 0; i < input->nvariables; ++i)
00531     {
00532         child = json_node_new (JSON_NODE_OBJECT);
00533         object = json_node_get_object (child);
00534         json_object_set_string_member (object, LABEL_NAME,
00535                                         input->variable[i].name);
00536         json_object_set_float (object, LABEL_MINIMUM,
00537                                 input->variable[i].rangemin);
00538         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00539             json_object_set_float (object,
LABEL_ABSOLUTE_MINIMUM,
00540                                     input->variable[i].rangeminabs);
00541         json_object_set_float (object, LABEL_MAXIMUM,
00542                                 input->variable[i].rangemax);
00543         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00544             json_object_set_float (object,
LABEL_ABSOLUTE_MAXIMUM,
00545                                     input->variable[i].rangemaxabs);
00546         if (input->variable[i].precision !=
DEFAULT_PRECISION)
00547             json_object_set_uint (object, LABEL_PRECISION,
input->variable[i].precision);
00548         if (input->algorithm == ALGORITHM_SWEEP
|| input->algorithm == ALGORITHM_ORTHOGONAL)
00549             json_object_set_uint (object, LABEL_NSWEEPS,
input->variable[i].nsweeps);
00550         else if (input->algorithm == ALGORITHM_GENETIC)
00551             json_object_set_uint (object, LABEL_NBITS,
input->variable[i].nbits);
00552         if (input->nsteps)
00553             json_object_set_float (object, LABEL_STEP,
input->variable[i].step);
00554         json_array_add_element (array, child);
00555     }
00556     json_object_set_array_member (object, LABEL_VARIABLES, array);
00557
00558     // Saving the error norm
00559     switch (input->norm)
00560     {
00561         case ERROR_NORM_MAXIMUM:
00562             json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00563             break;
00564         case ERROR_NORM_P:
00565             json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00566             json_object_set_float (object, LABEL_P, input->
p);
00567             break;
00568         case ERROR_NORM_TAXICAB:
00569             json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00570     }
00571
00572     #if DEBUG_INTERFACE
00573     fprintf (stderr, "input_save_json: end\n");
00574     #endif
00575 }
00576
00577 void
00578 input_save (char *filename)
00579 {
00580     xmlDoc *doc;
00581     JsonGenerator *generator;
00582
00583     #if DEBUG_INTERFACE
00584     fprintf (stderr, "input_save: start\n");
00585     #endif
00586
00587     // Getting the input file directory
00588     input->name = g_path_get_basename (filename);
00589     input->directory = g_path_get_dirname (filename);
00590
00591     if (input->type == INPUT_TYPE_XML)
00592     {
00593         // Opening the input file
00594         doc = xmlNewDoc ((const xmlChar *) "1.0");
00595     }

```

```

00601     input_save_xml (doc);
00602
00603     // Saving the XML file
00604     xmlSaveFormatFile (filename, doc, 1);
00605
00606     // Freeing memory
00607     xmlFreeDoc (doc);
00608 }
00609 else
00610 {
00611     // Opening the input file
00612     generator = json_generator_new ();
00613     json_generator_set_pretty (generator, TRUE);
00614     input_save_json (generator);
00615
00616     // Saving the JSON file
00617     json_generator_to_file (generator, filename, NULL);
00618
00619     // Freeing memory
00620     g_object_unref (generator);
00621 }
00622
00623 #if DEBUG_INTERFACE
00624     fprintf (stderr, "input_save: end\n");
00625 #endif
00626 }
00627
00631 void
00632 options_new ()
00633 {
00634     #if DEBUG_INTERFACE
00635         fprintf (stderr, "options_new: start\n");
00636     #endif
00637     options->label_seed = (GtkLabel *)
00638         gtk_label_new (_("Pseudo-random numbers generator seed"));
00639     options->spin_seed = (GtkSpinButton *)
00640         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00641     gtk_widget_set_tooltip_text
00642         (GTK_WIDGET (options->spin_seed),
00643          _("Seed to init the pseudo-random numbers generator"));
00644     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00645     options->label_threads = (GtkLabel *)
00646         gtk_label_new (_("Threads number for the stochastic algorithm"));
00647     options->spin_threads
00648         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00649     gtk_widget_set_tooltip_text
00650         (GTK_WIDGET (options->spin_threads),
00651          _("Number of threads to perform the calibration/optimization for "
00652            "the stochastic algorithm"));
00653     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00654     options->label_climbing = (GtkLabel *)
00655         gtk_label_new (_("Threads number for the hill climbing method"));
00656     options->spin_climbing =
00657         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00658     gtk_widget_set_tooltip_text
00659         (GTK_WIDGET (options->spin_climbing),
00660          _("Number of threads to perform the calibration/optimization for the "
00661            "hill climbing method"));
00662     gtk_spin_button_set_value (options->spin_climbing,
00663                               (gdouble) nthreads_climbing);
00664     options->grid = (GtkGrid *) gtk_grid_new ();
00665     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00667     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00668                     0, 1, 1, 1);
00669     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00670                     1, 1, 1, 1);
00671     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
00672                     1);
00673     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
00674                     1);
00675     gtk_widget_show_all (GTK_WIDGET (options->grid));
00676     options->dialog = (GtkDialog *)
00677         gtk_dialog_new_with_buttons (_("Options"),
00678                                     window->window,
00679                                     GTK_DIALOG_MODAL,
00680                                     _("_OK"), GTK_RESPONSE_OK,
00681                                     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00682     gtk_container_add
00683         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00684          GTK_WIDGET (options->grid));
00685     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00686     {
00687         input->seed
00688             = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);

```

```

00689     nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00690     nthreads_climbing
00691     = gtk_spin_button_get_value_as_int (options->spin_climbing);
00692 }
00693 gtk_widget_destroy (GTK_WIDGET (options->dialog));
00694 #if DEBUG_INTERFACE
00695 fprintf (stderr, "options_new: end\n");
00696 #endif
00697 }
00698
00702 void
00703 running_new ()
00704 {
00705     #if DEBUG_INTERFACE
00706     fprintf (stderr, "running_new: start\n");
00707     #endif
00708     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00709     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00710     running->grid = (GtkGrid *) gtk_grid_new ();
00711     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00712     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00713     running->dialog = (GtkDialog *)
00714         gtk_dialog_new_with_buttons (_("Calculating"),
00715                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00716     gtk_container_add (GTK_CONTAINER
00717                       (gtk_dialog_get_content_area (running->dialog)),
00718                       GTK_WIDGET (running->grid));
00719     gtk_spinner_start (running->spinner);
00720     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721     #if DEBUG_INTERFACE
00722     fprintf (stderr, "running_new: end\n");
00723     #endif
00724 }
00725
00731 unsigned int
00732 window_get_algorithm ()
00733 {
00734     unsigned int i;
00735     #if DEBUG_INTERFACE
00736     fprintf (stderr, "window_get_algorithm: start\n");
00737     #endif
00738     i = gtk_array_get_active (window->button_algorithm,
00739                             NALGORITHMS);
00739     #if DEBUG_INTERFACE
00740     fprintf (stderr, "window_get_algorithm: %u\n", i);
00741     fprintf (stderr, "window_get_algorithm: end\n");
00742     #endif
00743     return i;
00744 }
00745
00751 unsigned int
00752 window_get_climbing ()
00753 {
00754     unsigned int i;
00755     #if DEBUG_INTERFACE
00756     fprintf (stderr, "window_get_climbing: start\n");
00757     #endif
00758     i = gtk_array_get_active (window->button_climbing,
00759                             NCLIMBINGS);
00759     #if DEBUG_INTERFACE
00760     fprintf (stderr, "window_get_climbing: %u\n", i);
00761     fprintf (stderr, "window_get_climbing: end\n");
00762     #endif
00763     return i;
00764 }
00765
00771 unsigned int
00772 window_get_norm ()
00773 {
00774     unsigned int i;
00775     #if DEBUG_INTERFACE
00776     fprintf (stderr, "window_get_norm: start\n");
00777     #endif
00778     i = gtk_array_get_active (window->button_norm,
00779                             NNORMS);
00779     #if DEBUG_INTERFACE
00780     fprintf (stderr, "window_get_norm: %u\n", i);
00781     fprintf (stderr, "window_get_norm: end\n");
00782     #endif
00783     return i;
00784 }
00785
00789 void
00790 window_save_climbing ()
00791 {
00792     #if DEBUG_INTERFACE
00793     fprintf (stderr, "window_save_climbing: start\n");

```

```

00794 #endif
00795     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_climbing)))
00796     {
00797         input->nsteps = gtk_spin_button_get_value_as_int (window->
spin_steps);
00798         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
00799         switch (window_get_climbing ())
00800         {
00801             case CLIMBING_METHOD_COORDINATES:
00802                 input->climbing = CLIMBING_METHOD_COORDINATES;
00803                 break;
00804             default:
00805                 input->climbing = CLIMBING_METHOD_RANDOM;
00806                 input->nestimates
= gtk_spin_button_get_value_as_int (window->spin_estimates);
00807         }
00808     }
00809 }
00810 else
00811     input->nsteps = 0;
00812 #if DEBUG_INTERFACE
00813     fprintf (stderr, "window_save_climbing: end\n");
00814 #endif
00815 }
00816
00822 int
00823 window_save ()
00824 {
00825     GtkFileChooserDialog *dlg;
00826     GtkFileFilter *filter1, *filter2;
00827     char *buffer;
00828
00829 #if DEBUG_INTERFACE
00830     fprintf (stderr, "window_save: start\n");
00831 #endif
00832
00833     // Opening the saving dialog
00834     dlg = (GtkFileChooserDialog *)
00835         gtk_file_chooser_dialog_new (_("Save file"),
00836                                     window->window,
00837                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00838                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00839                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00840     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00841     buffer = g_build_filename (input->directory, input->name, NULL);
00842     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00843     g_free (buffer);
00844
00845     // Adding XML filter
00846     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00847     gtk_file_filter_set_name (filter1, "XML");
00848     gtk_file_filter_add_pattern (filter1, "*.xml");
00849     gtk_file_filter_add_pattern (filter1, "*.XML");
00850     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00851
00852     // Adding JSON filter
00853     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00854     gtk_file_filter_set_name (filter2, "JSON");
00855     gtk_file_filter_add_pattern (filter2, "*.json");
00856     gtk_file_filter_add_pattern (filter2, "*.JSON");
00857     gtk_file_filter_add_pattern (filter2, "*.js");
00858     gtk_file_filter_add_pattern (filter2, "*.JS");
00859     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861     if (input->type == INPUT_TYPE_XML)
00862         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00863     else
00864         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00865
00866     // If OK response then saving
00867     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00868     {
00869         // Setting input file type
00870         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00871         buffer = (char *) gtk_file_filter_get_name (filter1);
00872         if (!strcmp (buffer, "XML"))
00873             input->type = INPUT_TYPE_XML;
00874         else
00875             input->type = INPUT_TYPE_JSON;
00876
00877         // Adding properties to the root XML node
00878         input->simulator = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (window->button_simulator));
00879         if (gtk_toggle_button_get_active
(GTK_TOGGLE_BUTTON (window->check_evaluator)))
00880             input->evaluator = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (window->button_evaluator));
00881     }

```

```

00884     else
00885         input->evaluator = NULL;
00886     if (input->type == INPUT_TYPE_XML)
00887     {
00888         input->result
00889             = (char *) xmlStrdup ((const xmlChar *)
00890                                   gtk_entry_get_text (window->entry_result));
00891         input->variables
00892             = (char *) xmlStrdup ((const xmlChar *)
00893                                   gtk_entry_get_text (window->entry_variables));
00894     }
00895     else
00896     {
00897         input->result = g_strdup (gtk_entry_get_text (window->
00898 entry_result));
00899         input->variables =
00900             g_strdup (gtk_entry_get_text (window->entry_variables));
00901     }
00902     // Setting the algorithm
00903     switch (window_get_algorithm ())
00904     {
00905         case ALGORITHM_MONTE_CARLO:
00906             input->algorithm = ALGORITHM_MONTE_CARLO;
00907             input->nsimulations
00908                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00909             input->niterations
00910                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00911             input->tolerance = gtk_spin_button_get_value (window->
00912 spin_tolerance);
00913             input->nbest = gtk_spin_button_get_value_as_int (window->
00914 spin_bests);
00915             window_save_climbing ();
00916             break;
00917         case ALGORITHM_SWEEP:
00918             input->algorithm = ALGORITHM_SWEEP;
00919             input->niterations
00920                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921             input->tolerance = gtk_spin_button_get_value (window->
00922 spin_tolerance);
00923             input->nbest = gtk_spin_button_get_value_as_int (window->
00924 spin_bests);
00925             window_save_climbing ();
00926             break;
00927         case ALGORITHM_ORTHOGONAL:
00928             input->algorithm = ALGORITHM_ORTHOGONAL;
00929             input->niterations
00930                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00931             input->tolerance = gtk_spin_button_get_value (window->
00932 spin_tolerance);
00933             input->nbest = gtk_spin_button_get_value_as_int (window->
00934 spin_bests);
00935             window_save_climbing ();
00936             break;
00937         default:
00938             input->algorithm = ALGORITHM_GENETIC;
00939             input->nsimulations
00940                 = gtk_spin_button_get_value_as_int (window->spin_population);
00941             input->niterations
00942                 = gtk_spin_button_get_value_as_int (window->spin_generations);
00943             input->mutation_ratio
00944                 = gtk_spin_button_get_value (window->spin_mutation);
00945             input->reproduction_ratio
00946                 = gtk_spin_button_get_value (window->spin_reproduction);
00947             input->adaptation_ratio
00948                 = gtk_spin_button_get_value (window->spin_adaptation);
00949             break;
00950     }
00951     input->norm = window_get_norm ();
00952     input->p = gtk_spin_button_get_value (window->spin_p);
00953     input->threshold = gtk_spin_button_get_value (window->
00954 spin_threshold);
00955
00956     // Saving the XML file
00957     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00958     input_save (buffer);
00959
00960     // Closing and freeing memory
00961     g_free (buffer);
00962     gtk_widget_destroy (GTK_WIDGET (dlg));
00963
00964     #if DEBUG_INTERFACE
00965         fprintf (stderr, "window_save: end\n");
00966     #endif
00967     return 1;
00968 }
00969
00970 // Closing and freeing memory

```

```

00963  gtk_widget_destroy (GTK_WIDGET (dlg));
00964  #if DEBUG_INTERFACE
00965  fprintf (stderr, "window_save: end\n");
00966  #endif
00967  return 0;
00968  }
00969
00973  void
00974  window_run ()
00975  {
00976  unsigned int i;
00977  char *msg, *msg2, buffer[64], buffer2[64];
00978  #if DEBUG_INTERFACE
00979  fprintf (stderr, "window_run: start\n");
00980  #endif
00981  if (!window_save ())
00982  {
00983  #if DEBUG_INTERFACE
00984  fprintf (stderr, "window_run: end\n");
00985  #endif
00986  return;
00987  }
00988  running_new ();
00989  while (gtk_events_pending ())
00990  gtk_main_iteration ();
00991  optimize_open ();
00992  #if DEBUG_INTERFACE
00993  fprintf (stderr, "window_run: closing running dialog\n");
00994  #endif
00995  gtk_spinner_stop (running->spinner);
00996  gtk_widget_destroy (GTK_WIDGET (running->dialog));
00997  #if DEBUG_INTERFACE
00998  fprintf (stderr, "window_run: displaying results\n");
00999  #endif
01000  snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01001  msg2 = g_strdup (buffer);
01002  for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01003  {
01004  snprintf (buffer, 64, "%s = %s\n",
01005           input->variable[i].name, format[input->
01006           variable[i].precision]);
01007  snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008  msg = g_strconcat (msg2, buffer2, NULL);
01009  g_free (msg2);
01010  snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01011           optimize->calculation_time);
01012  msg = g_strconcat (msg2, buffer, NULL);
01013  g_free (msg2);
01014  show_message (_("Best result"), msg, INFO_TYPE);
01015  g_free (msg);
01016  #if DEBUG_INTERFACE
01017  fprintf (stderr, "window_run: freeing memory\n");
01018  #endif
01019  optimize_free ();
01020  #if DEBUG_INTERFACE
01021  fprintf (stderr, "window_run: end\n");
01022  #endif
01023  }
01024
01028  void
01029  window_help ()
01030  {
01031  char *buffer, *buffer2;
01032  #if DEBUG_INTERFACE
01033  fprintf (stderr, "window_help: start\n");
01034  #endif
01035  buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01036                             _("user-manual.pdf"), NULL);
01037  buffer = g_filename_to_uri (buffer2, NULL, NULL);
01038  g_free (buffer2);
01039  #if GTK_MINOR_VERSION >= 22
01040  gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01041  #else
01042  gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01043  #endif
01044  #if DEBUG_INTERFACE
01045  fprintf (stderr, "window_help: uri=%s\n", buffer);
01046  #endif
01047  g_free (buffer);
01048  #if DEBUG_INTERFACE
01049  fprintf (stderr, "window_help: end\n");
01050  #endif
01051  }
01052
01056  void
01057  window_about ()

```

```

01058 {
01059     static const gchar *authors[] = {
01060         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01061         "Borja Latorre Garcés <borja.latorre@csic.es>",
01062         NULL
01063     };
01064     #if DEBUG_INTERFACE
01065     fprintf (stderr, "window_about: start\n");
01066     #endif
01067     gtk_show_about_dialog
01068     (window->window,
01069      "program_name", "MPCOTool",
01070      "comments",
01071      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01072       "A software to perform calibrations or optimizations of empirical "
01073       "parameters"),
01074      "authors", authors,
01075      "translator-credits",
01076      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01077       "(english, french and spanish)\n"
01078       "Uğur Çayoğlu (german)",
01079      "version", "4.0.1",
01080      "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
01081      "logo", window->logo,
01082      "website", "https://github.com/jburguete/mpcotool",
01083      "license-type", GTK_LICENSE_BSD, NULL);
01084     #if DEBUG_INTERFACE
01085     fprintf (stderr, "window_about: end\n");
01086     #endif
01087 }
01088
01092 void
01093 window_update_climbing ()
01094 {
01095     #if DEBUG_INTERFACE
01096     fprintf (stderr, "window_update_climbing: start\n");
01097     #endif
01098     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01099     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_climbing)))
01100     {
01101         gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01102         gtk_widget_show (GTK_WIDGET (window->label_step));
01103         gtk_widget_show (GTK_WIDGET (window->spin_step));
01104     }
01105     switch (window_get_climbing ())
01106     {
01107         case CLIMBING_METHOD_COORDINATES:
01108             gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01109             gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01110             break;
01111         default:
01112             gtk_widget_show (GTK_WIDGET (window->label_estimates));
01113             gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01114     }
01115     #if DEBUG_INTERFACE
01116     fprintf (stderr, "window_update_climbing: end\n");
01117     #endif
01118 }
01119
01123 void
01124 window_update ()
01125 {
01126     unsigned int i;
01127     #if DEBUG_INTERFACE
01128     fprintf (stderr, "window_update: start\n");
01129     #endif
01130     gtk_widget_set_sensitive
01131     (GTK_WIDGET (window->button_evaluator),
01132      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01133      (window->check_evaluator)));
01134     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01135     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01136     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01137     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01138     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01139     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01140     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01142     gtk_widget_hide (GTK_WIDGET (window->label_population));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01144     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01146     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01148     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01149     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01150     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));

```



```

01151 gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01152 gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01153 gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01154 gtk_widget_hide (GTK_WIDGET (window->label_bits));
01155 gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01156 gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01157 gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01158 gtk_widget_hide (GTK_WIDGET (window->label_step));
01159 gtk_widget_hide (GTK_WIDGET (window->spin_step));
01160 gtk_widget_hide (GTK_WIDGET (window->label_p));
01161 gtk_widget_hide (GTK_WIDGET (window->spin_p));
01162 i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01163 switch (window_get_algorithm ())
01164 {
01165     case ALGORITHM_MONTE_CARLO:
01166         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01167         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01168         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01169         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01170         if (i > 1)
01171         {
01172             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01173             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01174             gtk_widget_show (GTK_WIDGET (window->label_bests));
01175             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01176         }
01177         window_update_climbing ();
01178         break;
01179     case ALGORITHM_SWEEP:
01180     case ALGORITHM_ORTHOGONAL:
01181         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01182         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01183         if (i > 1)
01184         {
01185             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01186             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01187             gtk_widget_show (GTK_WIDGET (window->label_bests));
01188             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01189         }
01190         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01191         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01192         gtk_widget_show (GTK_WIDGET (window->check_climbing));
01193         window_update_climbing ();
01194         break;
01195     default:
01196         gtk_widget_show (GTK_WIDGET (window->label_population));
01197         gtk_widget_show (GTK_WIDGET (window->spin_population));
01198         gtk_widget_show (GTK_WIDGET (window->label_generations));
01199         gtk_widget_show (GTK_WIDGET (window->spin_generations));
01200         gtk_widget_show (GTK_WIDGET (window->label_mutation));
01201         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01202         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01203         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01204         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01205         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01206         gtk_widget_show (GTK_WIDGET (window->label_bits));
01207         gtk_widget_show (GTK_WIDGET (window->spin_bits));
01208     }
01209     gtk_widget_set_sensitive
01210     (GTK_WIDGET (window->button_remove_experiment),
01211      input->nexperiments > 1);
01211     gtk_widget_set_sensitive
01212     (GTK_WIDGET (window->button_remove_variable), input->
01213      nvariables > 1);
01213     for (i = 0; i < input->experiment->ninputs; ++i)
01214     {
01215         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01216         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01217         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01218         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01219         g_signal_handler_block
01220         (window->check_template[i], window->id_template[i]);
01221         g_signal_handler_block (window->button_template[i], window->
01222         id_input[i]);
01222         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01223         (window->check_template[i]), 1);
01224         g_signal_handler_unblock (window->button_template[i],
01225         window->id_input[i]);
01226         g_signal_handler_unblock (window->check_template[i],
01227         window->id_template[i]);
01228     }
01229     if (i > 0)
01230     {
01231         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01232         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01233         gtk_toggle_button_get_active
01234         GTK_TOGGLE_BUTTON (window->check_template

```

```

01235                                     [i - 1]));
01236     }
01237     if (i < MAX_NINPUTS)
01238     {
01239         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01240         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01241         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01242         gtk_widget_set_sensitive
01243             (GTK_WIDGET (window->button_template[i]),
01244              gtk_toggle_button_get_active
01245                (GTK_TOGGLE_BUTTON (window->check_template[i]));
01246         g_signal_handler_block
01247             (window->check_template[i], window->id_template[i]);
01248         g_signal_handler_block (window->button_template[i], window->
01249             id_input[i]);
01249         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01250             (window->check_template[i]), 0);
01251         g_signal_handler_unblock (window->button_template[i],
01252             window->id_input[i]);
01253         g_signal_handler_unblock (window->check_template[i],
01254             window->id_template[i]);
01255     }
01256     while (++i < MAX_NINPUTS)
01257     {
01258         gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01259         gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01260     }
01261     gtk_widget_set_sensitive
01262         (GTK_WIDGET (window->spin_minabs),
01263          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01264     gtk_widget_set_sensitive
01265         (GTK_WIDGET (window->spin_maxabs),
01266          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01267     if (window_get_norm () == ERROR_NORM_P)
01268     {
01269         gtk_widget_show (GTK_WIDGET (window->label_p));
01270         gtk_widget_show (GTK_WIDGET (window->spin_p));
01271     }
01272     #if DEBUG_INTERFACE
01273     fprintf (stderr, "window_update: end\n");
01274     #endif
01275 }
01276
01280 void
01281 window_set_algorithm ()
01282 {
01283     int i;
01284     #if DEBUG_INTERFACE
01285     fprintf (stderr, "window_set_algorithm: start\n");
01286     #endif
01287     i = window_get_algorithm ();
01288     switch (i)
01289     {
01290     case ALGORITHM_SWEEP:
01291     case ALGORITHM_ORTHOGONAL:
01292         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01293         if (i < 0)
01294             i = 0;
01295         gtk_spin_button_set_value (window->spin_sweeps,
01296             (gdouble) input->variable[i].
01297             nsweeps);
01298         break;
01299     case ALGORITHM_GENETIC:
01300         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01301         if (i < 0)
01302             i = 0;
01303         gtk_spin_button_set_value (window->spin_bits,
01304             (gdouble) input->variable[i].nbits);
01305     }
01306     window_update ();
01307     #if DEBUG_INTERFACE
01308     fprintf (stderr, "window_set_algorithm: end\n");
01309     #endif
01310 }
01311
01314 void
01315 window_set_experiment ()
01316 {
01317     unsigned int i, j;
01318     char *buffer1, *buffer2;
01319     #if DEBUG_INTERFACE
01320     fprintf (stderr, "window_set_experiment: start\n");
01321     #endif
01322     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01323     gtk_spin_button_set_value (window->spin_weight, input->
01324         experiment[i].weight);
01325     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);

```

```

01325     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01326     g_free (buffer1);
01327     g_signal_handler_block
01328         (window->button_experiment, window->id_experiment_name);
01329     gtk_file_chooser_set_filename
01330         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01331     g_signal_handler_unblock
01332         (window->button_experiment, window->id_experiment_name);
01333     g_free (buffer2);
01334     for (j = 0; j < input->experiment->ninputs; ++j)
01335     {
01336         g_signal_handler_block (window->button_template[j], window->
01337             id_input[j]);
01338         buffer2 =
01339             g_build_filename (input->directory, input->experiment[i].
01340                 stencil[j],
01341                     NULL);
01342         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01343             (window->button_template[j]), buffer2);
01344         g_free (buffer2);
01345         g_signal_handler_unblock
01346             (window->button_template[j], window->id_input[j]);
01347     }
01348     #if DEBUG_INTERFACE
01349     fprintf (stderr, "window_set_experiment: end\n");
01350     #endif
01351 }
01352
01353 void
01354 window_remove_experiment ()
01355 {
01356     unsigned int i, j;
01357     #if DEBUG_INTERFACE
01358     fprintf (stderr, "window_remove_experiment: start\n");
01359     #endif
01360     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01361     g_signal_handler_block (window->combo_experiment, window->
01362         id_experiment);
01363     gtk_combo_box_text_remove (window->combo_experiment, i);
01364     g_signal_handler_unblock (window->combo_experiment, window->
01365         id_experiment);
01366     experiment_free (input->experiment + i, input->
01367         type);
01368     --input->nexperiments;
01369     for (j = i; j < input->nexperiments; ++j)
01370         memcpy (input->experiment + j, input->experiment + j + 1,
01371             sizeof (Experiment));
01372     j = input->nexperiments - 1;
01373     if (i > j)
01374         i = j;
01375     for (j = 0; j < input->experiment->ninputs; ++j)
01376         g_signal_handler_block (window->button_template[j], window->
01377             id_input[j]);
01378     g_signal_handler_block
01379         (window->button_experiment, window->id_experiment_name);
01380     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01381     g_signal_handler_unblock
01382         (window->button_experiment, window->id_experiment_name);
01383     for (j = 0; j < input->experiment->ninputs; ++j)
01384         g_signal_handler_unblock (window->button_template[j], window->
01385             id_input[j]);
01386     window_update ();
01387     #if DEBUG_INTERFACE
01388     fprintf (stderr, "window_remove_experiment: end\n");
01389     #endif
01390 }
01391
01392 void
01393 window_add_experiment ()
01394 {
01395     unsigned int i, j;
01396     #if DEBUG_INTERFACE
01397     fprintf (stderr, "window_add_experiment: start\n");
01398     #endif
01399     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01400     g_signal_handler_block (window->combo_experiment, window->
01401         id_experiment);
01402     gtk_combo_box_text_insert_text
01403         (window->combo_experiment, i, input->experiment[i].
01404             name);
01405     g_signal_handler_unblock (window->combo_experiment, window->
01406         id_experiment);
01407     input->experiment = (Experiment *) g_realloc
01408         (input->experiment, (input->nexperiments + 1) * sizeof (
01409             Experiment));
01410     for (j = input->nexperiments - 1; j > i; --j)
01411         memcpy (input->experiment + j + 1, input->experiment + j,

```

```

01407         sizeof (Experiment));
01408     input->experiment[j + 1].weight = input->experiment[j].
weight;
01409     input->experiment[j + 1].ninputs = input->
experiment[j].ninputs;
01410     if (input->type == INPUT_TYPE_XML)
01411     {
01412         input->experiment[j + 1].name
01413         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
name);
01414         for (j = 0; j < input->experiment->ninputs; ++j)
01415             input->experiment[i + 1].stencil[j]
01416             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
stencil[j]);
01417     }
01418     else
01419     {
01420         input->experiment[j + 1].name = g_strdup (input->
experiment[j].name);
01421         for (j = 0; j < input->experiment->ninputs; ++j)
01422             input->experiment[i + 1].stencil[j]
01423             = g_strdup (input->experiment[i].stencil[j]);
01424     }
01425     ++input->nexperiments;
01426     for (j = 0; j < input->experiment->ninputs; ++j)
01427         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01428     g_signal_handler_block
01429     (window->button_experiment, window->id_experiment_name);
01430     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01431     g_signal_handler_unblock
01432     (window->button_experiment, window->id_experiment_name);
01433     for (j = 0; j < input->experiment->ninputs; ++j)
01434         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
01435     window_update ();
01436 #if DEBUG_INTERFACE
01437     fprintf (stderr, "window_add_experiment: end\n");
01438 #endif
01439 }
01440
01441 void
01442 window_name_experiment ()
01443 {
01444     unsigned int i;
01445     char *buffer;
01446     GFile *file1, *file2;
01447 #if DEBUG_INTERFACE
01448     fprintf (stderr, "window_name_experiment: start\n");
01449 #endif
01450     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01451     file1
01452     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01453     file2 = g_file_new_for_path (input->directory);
01454     buffer = g_file_get_relative_path (file2, file1);
01455     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01456     gtk_combo_box_text_remove (window->combo_experiment, i);
01457     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01458     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01459     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01460     g_free (buffer);
01461     g_object_unref (file2);
01462     g_object_unref (file1);
01463 #if DEBUG_INTERFACE
01464     fprintf (stderr, "window_name_experiment: end\n");
01465 #endif
01466 }
01467
01468 void
01469 window_weight_experiment ()
01470 {
01471     unsigned int i;
01472 #if DEBUG_INTERFACE
01473     fprintf (stderr, "window_weight_experiment: start\n");
01474 #endif
01475     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01476     input->experiment[i].weight = gtk_spin_button_get_value (window->
spin_weight);
01477 #if DEBUG_INTERFACE
01478     fprintf (stderr, "window_weight_experiment: end\n");
01479 #endif
01480 }
01481
01482 void
01483 window_inputs_experiment ()

```

```

01493 {
01494     unsigned int j;
01495     #if DEBUG_INTERFACE
01496     fprintf (stderr, "window_inputs_experiment: start\n");
01497     #endif
01498     j = input->experiment->ninputs - 1;
01499     if (j
01500         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01501             (window->check_template[j])))
01502         --input->experiment->ninputs;
01503     if (input->experiment->ninputs < MAX_NINPUTS
01504         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01505             (window->check_template[j])))
01506         ++input->experiment->ninputs;
01507     window_update ();
01508     #if DEBUG_INTERFACE
01509     fprintf (stderr, "window_inputs_experiment: end\n");
01510     #endif
01511 }
01512
01516 void
01517 window_template_experiment (void *data)
01518 {
01519     unsigned int i, j;
01520     char *buffer;
01521     GFile *file1, *file2;
01522     #if DEBUG_INTERFACE
01523     fprintf (stderr, "window_template_experiment: start\n");
01524     #endif
01525     i = (size_t) data;
01526     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01527     file1
01528         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01529     file2 = g_file_new_for_path (input->directory);
01530     buffer = g_file_get_relative_path (file2, file1);
01531     if (input->type == INPUT_TYPE_XML)
01532         input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01533     else
01534         input->experiment[j].stencil[i] = g_strdup (buffer);
01535     g_free (buffer);
01536     g_object_unref (file2);
01537     g_object_unref (file1);
01538     #if DEBUG_INTERFACE
01539     fprintf (stderr, "window_template_experiment: end\n");
01540     #endif
01541 }
01542
01543 void
01544 window_set_variable ()
01545 {
01546     unsigned int i;
01547     #if DEBUG_INTERFACE
01548     fprintf (stderr, "window_set_variable: start\n");
01549     #endif
01550     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01551     g_signal_handler_block (window->entry_variable, window->
01552         id_variable_label);
01553     gtk_entry_set_text (window->entry_variable, input->variable[i].
01554         name);
01555     g_signal_handler_unblock (window->entry_variable, window->
01556         id_variable_label);
01557     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01558         rangemin);
01559     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01560         rangemax);
01561     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01562     {
01563         gtk_spin_button_set_value (window->spin_minabs,
01564             input->variable[i].rangeminabs);
01565         gtk_toggle_button_set_active
01566             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01567     }
01568     else
01569     {
01570         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01571         gtk_toggle_button_set_active
01572             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01573     }
01574     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01575     {
01576         gtk_spin_button_set_value (window->spin_maxabs,
01577             input->variable[i].rangemaxabs);
01578         gtk_toggle_button_set_active
01579             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01580     }
01581     else
01582     {

```

```

01582     gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01583     gtk_toggle_button_set_active
01584     (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01585 }
01586 gtk_spin_button_set_value (window->spin_precision,
01587     input->variable[i].precision);
01588 gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
01589 if (input->nsteps)
01590     gtk_spin_button_set_value (window->spin_step, input->variable[i].
step);
01591 #if DEBUG_INTERFACE
01592     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01593         input->variable[i].precision);
01594 #endif
01595     switch (window_get_algorithm ())
01596     {
01597     case ALGORITHM_SWEEP:
01598     case ALGORITHM_ORTHOGONAL:
01599         gtk_spin_button_set_value (window->spin_sweeps,
01600             (gdouble) input->variable[i].
nsteps);
01601 #if DEBUG_INTERFACE
01602         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01603             input->variable[i].nsweeps);
01604 #endif
01605         break;
01606     case ALGORITHM_GENETIC:
01607         gtk_spin_button_set_value (window->spin_bits,
01608             (gdouble) input->variable[i].nbits);
01609 #if DEBUG_INTERFACE
01610         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01611             input->variable[i].nbits);
01612 #endif
01613         break;
01614     }
01615     window_update ();
01616 #if DEBUG_INTERFACE
01617     fprintf (stderr, "window_set_variable: end\n");
01618 #endif
01619 }
01620
01624 void
01625 window_remove_variable ()
01626 {
01627     unsigned int i, j;
01628     #if DEBUG_INTERFACE
01629         fprintf (stderr, "window_remove_variable: start\n");
01630     #endif
01631     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01632     g_signal_handler_block (window->combo_variable, window->
id_variable);
01633     gtk_combo_box_text_remove (window->combo_variable, i);
01634     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01635     xmlFree (input->variable[i].name);
01636     --input->nvariables;
01637     for (j = i; j < input->nvariables; ++j)
01638         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01639     j = input->nvariables - 1;
01640     if (i > j)
01641         i = j;
01642     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01643     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01644     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01645     window_update ();
01646 #if DEBUG_INTERFACE
01647     fprintf (stderr, "window_remove_variable: end\n");
01648 #endif
01649 }
01650
01654 void
01655 window_add_variable ()
01656 {
01657     unsigned int i, j;
01658     #if DEBUG_INTERFACE
01659         fprintf (stderr, "window_add_variable: start\n");
01660     #endif
01661     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01662     g_signal_handler_block (window->combo_variable, window->
id_variable);
01663     gtk_combo_box_text_insert_text (window->combo_variable, i,
01664         input->variable[i].name);
01665     g_signal_handler_unblock (window->combo_variable, window->

```

```

    id_variable);
01666     input->variable = (Variable *) g_realloc
01667     (input->variable, (input->nvariables + 1) * sizeof (
    Variable));
01668     for (j = input->nvariables - 1; j > i; --j)
01669         memcpy (input->variable + j + 1, input->variable + j, sizeof (
    Variable));
01670     memcpy (input->variable + j + 1, input->variable + j, sizeof (
    Variable));
01671     if (input->type == INPUT_TYPE_XML)
01672         input->variable[j + 1].name
01673         = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01674     else
01675         input->variable[j + 1].name = g_strdup (input->
    variable[j].name);
01676     ++input->nvariables;
01677     g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
01678     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01679     g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
01680     window_update ();
01681     #if DEBUG_INTERFACE
01682     fprintf (stderr, "window_add_variable: end\n");
01683     #endif
01684 }
01685
01686 void
01687 window_label_variable ()
01688 {
01689     unsigned int i;
01690     const char *buffer;
01691     #if DEBUG_INTERFACE
01692     fprintf (stderr, "window_label_variable: start\n");
01693     #endif
01694     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01695     buffer = gtk_entry_get_text (window->entry_variable);
01696     g_signal_handler_block (window->combo_variable, window->
    id_variable);
01697     gtk_combo_box_text_remove (window->combo_variable, i);
01698     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01699     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01700     g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
01701     #if DEBUG_INTERFACE
01702     fprintf (stderr, "window_label_variable: end\n");
01703     #endif
01704 }
01705
01706 void
01707 window_precision_variable ()
01708 {
01709     unsigned int i;
01710     #if DEBUG_INTERFACE
01711     fprintf (stderr, "window_precision_variable: start\n");
01712     #endif
01713     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01714     input->variable[i].precision
01715     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01716     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
    precision);
01717     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
    precision);
01718     gtk_spin_button_set_digits (window->spin_minabs,
    input->variable[i].precision);
01719     gtk_spin_button_set_digits (window->spin_maxabs,
    input->variable[i].precision);
01720     #if DEBUG_INTERFACE
01721     fprintf (stderr, "window_precision_variable: end\n");
01722     #endif
01723 }
01724
01725 void
01726 window_rangemin_variable ()
01727 {
01728     unsigned int i;
01729     #if DEBUG_INTERFACE
01730     fprintf (stderr, "window_rangemin_variable: start\n");
01731     #endif
01732     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01733     input->variable[i].rangemin = gtk_spin_button_get_value (window->
    spin_min);
01734     #if DEBUG_INTERFACE
01735     fprintf (stderr, "window_rangemin_variable: end\n");
01736     #endif
01737 }
01738
01739
01740

```

```

01753 void
01754 window_rangemax_variable ()
01755 {
01756     unsigned int i;
01757     #if DEBUG_INTERFACE
01758     fprintf (stderr, "window_rangemax_variable: start\n");
01759     #endif
01760     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01761     input->variable[i].rangemax = gtk_spin_button_get_value (window->
spin_max);
01762     #if DEBUG_INTERFACE
01763     fprintf (stderr, "window_rangemax_variable: end\n");
01764     #endif
01765 }
01766
01770 void
01771 window_rangeminabs_variable ()
01772 {
01773     unsigned int i;
01774     #if DEBUG_INTERFACE
01775     fprintf (stderr, "window_rangeminabs_variable: start\n");
01776     #endif
01777     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01778     input->variable[i].rangeminabs
= gtk_spin_button_get_value (window->spin_minabs);
01779     #if DEBUG_INTERFACE
01780     fprintf (stderr, "window_rangeminabs_variable: end\n");
01781     #endif
01782 }
01783
01784
01788 void
01789 window_rangemaxabs_variable ()
01790 {
01791     unsigned int i;
01792     #if DEBUG_INTERFACE
01793     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01794     #endif
01795     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01796     input->variable[i].rangemaxabs
= gtk_spin_button_get_value (window->spin_maxabs);
01797     #if DEBUG_INTERFACE
01798     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01799     #endif
01800 }
01801
01802
01806 void
01807 window_step_variable ()
01808 {
01809     unsigned int i;
01810     #if DEBUG_INTERFACE
01811     fprintf (stderr, "window_step_variable: start\n");
01812     #endif
01813     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01814     input->variable[i].step = gtk_spin_button_get_value (window->
spin_step);
01815     #if DEBUG_INTERFACE
01816     fprintf (stderr, "window_step_variable: end\n");
01817     #endif
01818 }
01819
01823 void
01824 window_update_variable ()
01825 {
01826     int i;
01827     #if DEBUG_INTERFACE
01828     fprintf (stderr, "window_update_variable: start\n");
01829     #endif
01830     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01831     if (i < 0)
01832         i = 0;
01833     switch (window_get_algorithm ())
01834     {
01835         case ALGORITHM_SWEEP:
01836         case ALGORITHM_ORTHOGONAL:
01837             input->variable[i].nsweeps
= gtk_spin_button_get_value_as_int (window->spin_sweeps);
01838             #if DEBUG_INTERFACE
01839             fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
input->variable[i].nsweeps);
01840             #endif
01841             break;
01842         case ALGORITHM_GENETIC:
01843             input->variable[i].nbits
= gtk_spin_button_get_value_as_int (window->spin_bits);
01844             #if DEBUG_INTERFACE
01845             fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
input->variable[i].nbits);
01846             #endif
01847     }
01848 }

```



```

01850 #endif
01851     }
01852 #if DEBUG_INTERFACE
01853     fprintf (stderr, "window_update_variable: end\n");
01854 #endif
01855 }
01856
01857 int
01863 window_read (char *filename)
01864 {
01865     unsigned int i;
01866     char *buffer;
01867 #if DEBUG_INTERFACE
01868     fprintf (stderr, "window_read: start\n");
01869 #endif
01870
01871     // Reading new input file
01872     input_free ();
01873     input->result = input->variables = NULL;
01874     if (!input_open (filename))
01875     {
01876 #if DEBUG_INTERFACE
01877         fprintf (stderr, "window_read: end\n");
01878 #endif
01879         return 0;
01880     }
01881
01882     // Setting GTK+ widgets data
01883     gtk_entry_set_text (window->entry_result, input->result);
01884     gtk_entry_set_text (window->entry_variables, input->
variables);
01885     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01886     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01887     g_free (buffer);
01888     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01889
01890     if (input->evaluator)
01891     {
01892         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01893         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01894         g_free (buffer);
01895     }
01896     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01900     switch (input->algorithm)
01901     {
01902     case ALGORITHM_MONTE_CARLO:
01903         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01904
01905         // fallthrough
01906     case ALGORITHM_SWEEP:
01907     case ALGORITHM_ORTHOGONAL:
01908         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01909         gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01910         gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01911         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_climbing),
input->nsteps);
01912         if (input->nsteps)
01913         {
01914             gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_climbing[input->
climbing]),
TRUE);
01915             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01916             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01917
01918             switch (input->climbing)
01919             {
01920             case CLIMBING_METHOD_RANDOM:
01921                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01922             }
01923         }
01924         break;
01925     default:
01926         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
01927     }
01928 }
01929
01930 break;
01931 default:
01932     gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
01933 }

```

```

01934         gtk_spin_button_set_value (window->spin_generations,
01935                                     (gdouble) input->niterations);
01936         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01937         gtk_spin_button_set_value (window->spin_reproduction,
01938                                     input->reproduction_ratio);
01939         gtk_spin_button_set_value (window->spin_adaptation,
01940                                     input->adaptation_ratio);
01941     }
01942     gtk_toggle_button_set_active
01943     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01944     gtk_spin_button_set_value (window->spin_p, input->p);
01945     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01946     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01947     g_signal_handler_block (window->button_experiment,
01948                             window->id_experiment_name);
01949     gtk_combo_box_text_remove_all (window->combo_experiment);
01950     for (i = 0; i < input->nexperiments; ++i)
01951         gtk_combo_box_text_append_text (window->combo_experiment,
01952                                         input->experiment[i].name);
01953     g_signal_handler_unblock
01954     (window->button_experiment, window->id_experiment_name);
01955     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01956     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01957     g_signal_handler_block (window->combo_variable, window->
id_variable);
01958     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01959     gtk_combo_box_text_remove_all (window->combo_variable);
01960     for (i = 0; i < input->nvariables; ++i)
01961         gtk_combo_box_text_append_text (window->combo_variable,
01962                                         input->variable[i].name);
01963     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01964     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01965     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01966     window_set_variable ();
01967     window_update ();
01968
01969     #if DEBUG_INTERFACE
01970     fprintf (stderr, "window_read: end\n");
01971     #endif
01972     return 1;
01973 }
01974
01978 void
01979 window_open ()
01980 {
01981     GtkFileChooserDialog *dlg;
01982     GtkFileFilter *filter;
01983     char *buffer, *directory, *name;
01984
01985     #if DEBUG_INTERFACE
01986     fprintf (stderr, "window_open: start\n");
01987     #endif
01988
01989     // Saving a backup of the current input file
01990     directory = g_strdup (input->directory);
01991     name = g_strdup (input->name);
01992
01993     // Opening dialog
01994     dlg = (GtkFileChooserDialog *)
01995         gtk_file_chooser_dialog_new (_("Open input file"),
01996                                     window->window,
01997                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01998                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
01999                                     _("_OK"), GTK_RESPONSE_OK, NULL);
02000
02001     // Adding XML filter
02002     filter = (GtkFileFilter *) gtk_file_filter_new ();
02003     gtk_file_filter_set_name (filter, "XML");
02004     gtk_file_filter_add_pattern (filter, "*.xml");
02005     gtk_file_filter_add_pattern (filter, "*.XML");
02006     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02007
02008     // Adding JSON filter
02009     filter = (GtkFileFilter *) gtk_file_filter_new ();
02010     gtk_file_filter_set_name (filter, "JSON");
02011     gtk_file_filter_add_pattern (filter, "*.json");
02012     gtk_file_filter_add_pattern (filter, "*.JSON");
02013     gtk_file_filter_add_pattern (filter, "*.js");
02014     gtk_file_filter_add_pattern (filter, "*.JS");
02015     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

```

```

02016
02017 // If OK saving
02018 while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02019 {
02020
02021     // Trying to open the input file
02022     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02023     if (!window_read (buffer))
02024     {
02025 #if DEBUG_INTERFACE
02026         fprintf (stderr, "window_open: error reading input file\n");
02027 #endif
02028         g_free (buffer);
02029
02030         // Reading backup file on error
02031         buffer = g_build_filename (directory, name, NULL);
02032         input->result = input->variables = NULL;
02033         if (!input_open (buffer))
02034         {
02035
02036             // Closing on backup file reading error
02037 #if DEBUG_INTERFACE
02038             fprintf (stderr, "window_read: error reading backup file\n");
02039 #endif
02040             g_free (buffer);
02041             break;
02042         }
02043         g_free (buffer);
02044     }
02045     else
02046     {
02047         g_free (buffer);
02048         break;
02049     }
02050 }
02051
02052 // Freeing and closing
02053 g_free (name);
02054 g_free (directory);
02055 gtk_widget_destroy (GTK_WIDGET (dlg));
02056 #if DEBUG_INTERFACE
02057     fprintf (stderr, "window_open: end\n");
02058 #endif
02059 }
02060
02061 void
02062 window_new (GtkApplication * application)
02063 {
02064     unsigned int i;
02065     char *buffer, *buffer2, buffer3[64];
02066     char *label_algorithm[NALGORITHMS] = {
02067         "_Monte-Carlo", "_Sweep", "_Genetic", "_Orthogonal"
02068     };
02069     char *tip_algorithm[NALGORITHMS] = {
02070         _("Monte-Carlo brute force algorithm"),
02071         _("Sweep brute force algorithm"),
02072         _("Genetic algorithm"),
02073         _("Orthogonal sampling brute force algorithm"),
02074     };
02075     char *label_climbing[NCLIMBINGS] = {
02076         _("_Coordinates climbing"), _("_Random climbing")
02077     };
02078     char *tip_climbing[NCLIMBINGS] = {
02079         _("Coordinates climbing estimate method"),
02080         _("Random climbing estimate method")
02081     };
02082     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02083     char *tip_norm[NNORMS] = {
02084         _("Euclidean error norm (L2)"),
02085         _("Maximum error norm (L)"),
02086         _("P error norm (Lp)"),
02087         _("Taxicab error norm (L1)")
02088     };
02089
02090 #if DEBUG_INTERFACE
02091     fprintf (stderr, "window_new: start\n");
02092 #endif
02093
02094 // Creating the window
02095 window->window = main_window
02096     = (GtkWindow *) gtk_application_window_new (application);
02097
02098 // Finish when closing the window
02099 g_signal_connect_swapped (window->window, "delete-event",
02100     G_CALLBACK (g_application_quit),
02101     G_APPLICATION (application));
02102
02103
02104
02105

```

```

02106 // Setting the window title
02107 gtk_window_set_title (window->window, "MPCOTool");
02108
02109 // Creating the open button
02110 window->button_open = (GtkToolButton *) gtk_tool_button_new
02111     (gtk_image_new_from_icon_name ("document-open",
02112         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02113 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02114
02115 // Creating the save button
02116 window->button_save = (GtkToolButton *) gtk_tool_button_new
02117     (gtk_image_new_from_icon_name ("document-save",
02118         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02119 g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
02120     NULL);
02121
02122 // Creating the run button
02123 window->button_run = (GtkToolButton *) gtk_tool_button_new
02124     (gtk_image_new_from_icon_name ("system-run",
02125         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02126 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02127
02128 // Creating the options button
02129 window->button_options = (GtkToolButton *) gtk_tool_button_new
02130     (gtk_image_new_from_icon_name ("preferences-system",
02131         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02132 g_signal_connect (window->button_options, "clicked", options_new, NULL);
02133
02134 // Creating the help button
02135 window->button_help = (GtkToolButton *) gtk_tool_button_new
02136     (gtk_image_new_from_icon_name ("help-browser",
02137         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02138 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02139
02140 // Creating the about button
02141 window->button_about = (GtkToolButton *) gtk_tool_button_new
02142     (gtk_image_new_from_icon_name ("help-about",
02143         GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02144 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02145
02146 // Creating the exit button
02147 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02148     (gtk_image_new_from_icon_name ("application-exit",
02149         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02150 g_signal_connect_swapped (window->button_exit, "clicked",
02151     G_CALLBACK (g_application_quit),
02152     G_APPLICATION (application));
02153
02154 // Creating the buttons bar
02155 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02156 gtk_toolbar_insert
02157     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02158 gtk_toolbar_insert
02159     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02160 gtk_toolbar_insert
02161     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02162 gtk_toolbar_insert
02163     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02164 gtk_toolbar_insert
02165     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02166 gtk_toolbar_insert
02167     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02168 gtk_toolbar_insert
02169     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02170 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02171
02172 // Creating the simulator program label and entry
02173 window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02174 window->button_simulator = (GtkFileChooserButton *)
02175     gtk_file_chooser_button_new (_("Simulator program"),
02176         GTK_FILE_CHOOSER_ACTION_OPEN);
02177 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02178     _("Simulator program executable file"));
02179 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02180
02181 // Creating the evaluator program label and entry
02182 window->check_evaluator = (GtkCheckButton *)
02183     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02184 g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
02185 window->button_evaluator = (GtkFileChooserButton *)
02186     gtk_file_chooser_button_new (_("Evaluator program"),
02187         GTK_FILE_CHOOSER_ACTION_OPEN);
02188 gtk_widget_set_tooltip_text
02189     (GTK_WIDGET (window->button_evaluator),
02190     _("Optional evaluator program executable file"));

```

```

02191
02192 // Creating the results files labels and entries
02193 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02194 window->entry_result = (GtkEntry *) gtk_entry_new ();
02195 gtk_widget_set_tooltip_text
02196 (GTK_WIDGET (window->entry_result), _("Best results file"));
02197 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02198 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02199 gtk_widget_set_tooltip_text
02200 (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02201
02202 // Creating the files grid and attaching widgets
02203 window->grid_files = (GtkGrid *) gtk_grid_new ();
02204 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02205                 0, 0, 1, 1);
02206 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02207                 1, 0, 1, 1);
02208 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02209                 0, 1, 1, 1);
02210 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02211                 1, 1, 1, 1);
02212 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02213                 0, 2, 1, 1);
02214 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02215                 1, 2, 1, 1);
02216 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02217                 0, 3, 1, 1);
02218 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02219                 1, 3, 1, 1);
02220
02221 // Creating the algorithm properties
02222 window->label_simulations = (GtkLabel *) gtk_label_new
02223 (_("Simulations number"));
02224 window->spin_simulations
02225 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02226 gtk_widget_set_tooltip_text
02227 (GTK_WIDGET (window->spin_simulations),
02228  _("Number of simulations to perform for each iteration"));
02229 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02230 window->label_iterations = (GtkLabel *)
02231 gtk_label_new (_("Iterations number"));
02232 window->spin_iterations
02233 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02234 gtk_widget_set_tooltip_text
02235 (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02236 g_signal_connect
02237 (window->spin_iterations, "value-changed", window_update, NULL);
02238 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02239 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02240 window->spin_tolerance =
02241 (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02242 gtk_widget_set_tooltip_text
02243 (GTK_WIDGET (window->spin_tolerance),
02244  _("Tolerance to set the variable interval on the next iteration"));
02245 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02246 window->spin_bests
02247 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02248 gtk_widget_set_tooltip_text
02249 (GTK_WIDGET (window->spin_bests),
02250  _("Number of best simulations used to set the variable interval "
02251    "on the next iteration"));
02252 window->label_population
02253 = (GtkLabel *) gtk_label_new (_("Population number"));
02254 window->spin_population
02255 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02256 gtk_widget_set_tooltip_text
02257 (GTK_WIDGET (window->spin_population),
02258  _("Number of population for the genetic algorithm"));
02259 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02260 window->label_generations
02261 = (GtkLabel *) gtk_label_new (_("Generations number"));
02262 window->spin_generations
02263 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02264 gtk_widget_set_tooltip_text
02265 (GTK_WIDGET (window->spin_generations),
02266  _("Number of generations for the genetic algorithm"));
02267 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02268 window->spin_mutation
02269 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);

```

```

02270 gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_mutation),
02272      _("Ratio of mutation for the genetic algorithm"));
02273 window->label_reproduction
02274     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02275 window->spin_reproduction
02276     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02277 gtk_widget_set_tooltip_text
02278     (GTK_WIDGET (window->spin_reproduction),
02279      _("Ratio of reproduction for the genetic algorithm"));
02280 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02281 window->spin_adaptation
02282     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02283 gtk_widget_set_tooltip_text
02284     (GTK_WIDGET (window->spin_adaptation),
02285      _("Ratio of adaptation for the genetic algorithm"));
02286 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02287 window->spin_threshold = (GtkSpinButton *)
02288     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02289                                     precision[DEFAULT_PRECISION]);
02289
02290 gtk_widget_set_tooltip_text
02291     (GTK_WIDGET (window->spin_threshold),
02292      _("Threshold in the objective function to finish the simulations"));
02293 window->scrolled_threshold =
02294     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02295 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02296                   GTK_WIDGET (window->spin_threshold));
02297 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02298 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02299 //                          GTK_ALIGN_FILL);
02300
02301 // Creating the hill climbing method properties
02302 window->check_climbing = (GtkCheckButton *)
02303     gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
02304 g_signal_connect (window->check_climbing, "clicked",
02305                  window_update, NULL);
02306 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02307 window->button_climbing[0] = (GtkRadioButton *)
02308     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02309 gtk_grid_attach (window->grid_climbing,
02310                 GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02311 g_signal_connect (window->button_climbing[0], "clicked",
02312                  window_update, NULL);
02313 for (i = 0; ++i < NCLIMBINGS;)
02314 {
02315     window->button_climbing[i] = (GtkRadioButton *)
02316         gtk_radio_button_new_with_mnemonic
02317         (gtk_radio_button_get_group (window->button_climbing[0]),
02318          label_climbing[i]);
02319     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02320                                 tip_climbing[i]);
02321     gtk_grid_attach (window->grid_climbing,
02322                     GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02323     g_signal_connect (window->button_climbing[i], "clicked",
02324                      window_update,
02325                      NULL);
02326 }
02327 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02328 window->spin_steps = (GtkSpinButton *)
02329     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02330 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02331 window->label_estimates
02332     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02333 window->spin_estimates = (GtkSpinButton *)
02334     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02335 window->label_relaxation
02336     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02337 window->spin_relaxation = (GtkSpinButton *)
02338     gtk_spin_button_new_with_range (0., 2., 0.001);
02339 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
02340 label_steps),
02341                 0, NCLIMBINGS, 1, 1);
02342 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
02343 spin_steps),
02344                 1, NCLIMBINGS, 1, 1);
02345 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
02346 label_estimates),
02347                 0, NCLIMBINGS + 1, 1, 1);
02348 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
02349 spin_estimates),
02350                 1, NCLIMBINGS + 1, 1, 1);
02351 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
02352 label_relaxation),
02353                 0, NCLIMBINGS + 2, 1, 1);
02354 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
02355 spin_relaxation),
02356                 1, NCLIMBINGS + 2, 1, 1);

```

```

02348
02349 // Creating the array of algorithms
02350 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02351 window->button_algorithm[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02353 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02354     tip_algorithm[0]);
02355 gtk_grid_attach (window->grid_algorithm,
02356     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02357 g_signal_connect (window->button_algorithm[0], "clicked",
02358     window_set_algorithm, NULL);
02359 for (i = 0; ++i < NALGORITHMS;)
02360 {
02361     window->button_algorithm[i] = (GtkRadioButton *)
02362         gtk_radio_button_new_with_mnemonic
02363         (gtk_radio_button_get_group (window->button_algorithm[0]),
02364             label_algorithm[i]);
02365     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02366         tip_algorithm[i]);
02367     gtk_grid_attach (window->grid_algorithm,
02368         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02369     g_signal_connect (window->button_algorithm[i], "clicked",
02370         window_set_algorithm, NULL);
02371 }
02372 gtk_grid_attach (window->grid_algorithm,
02373     GTK_WIDGET (window->label_simulations),
02374     0, NALGORITHMS, 1, 1);
02375 gtk_grid_attach (window->grid_algorithm,
02376     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02377 gtk_grid_attach (window->grid_algorithm,
02378     GTK_WIDGET (window->label_iterations),
02379     0, NALGORITHMS + 1, 1, 1);
02380 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_iterations),
02381     1, NALGORITHMS + 1, 1, 1);
02382 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_tolerance),
02383     0, NALGORITHMS + 2, 1, 1);
02384 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_tolerance),
02385     1, NALGORITHMS + 2, 1, 1);
02386 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_bests),
02387     0, NALGORITHMS + 3, 1, 1);
02388 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_bests),
02389     1, NALGORITHMS + 3, 1, 1);
02390 gtk_grid_attach (window->grid_algorithm,
02391     GTK_WIDGET (window->label_population),
02392     0, NALGORITHMS + 4, 1, 1);
02393 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_population),
02394     1, NALGORITHMS + 4, 1, 1);
02395 gtk_grid_attach (window->grid_algorithm,
02396     GTK_WIDGET (window->label_generations),
02397     0, NALGORITHMS + 5, 1, 1);
02398 gtk_grid_attach (window->grid_algorithm,
02399     GTK_WIDGET (window->spin_generations),
02400     1, NALGORITHMS + 5, 1, 1);
02401 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_mutation),
02402     0, NALGORITHMS + 6, 1, 1);
02403 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_mutation),
02404     1, NALGORITHMS + 6, 1, 1);
02405 gtk_grid_attach (window->grid_algorithm,
02406     GTK_WIDGET (window->label_reproduction),
02407     0, NALGORITHMS + 7, 1, 1);
02408 gtk_grid_attach (window->grid_algorithm,
02409     GTK_WIDGET (window->spin_reproduction),
02410     1, NALGORITHMS + 7, 1, 1);
02411 gtk_grid_attach (window->grid_algorithm,
02412     GTK_WIDGET (window->label_adaptation),
02413     0, NALGORITHMS + 8, 1, 1);
02414 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_adaptation),
02415     1, NALGORITHMS + 8, 1, 1);
02416 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
check_climbing),
02417     0, NALGORITHMS + 9, 2, 1);
02418 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
grid_climbing),
02419     0, NALGORITHMS + 10, 2, 1);
02420 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_threshold),
02421     0, NALGORITHMS + 11, 1, 1);
02422 gtk_grid_attach (window->grid_algorithm,

```



```

02423         GTK_WIDGET (window->scrolled_threshold),
02424         1, NALGORITHMS + 11, 1, 1);
02425 window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02426 gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02427         GTK_WIDGET (window->grid_algorithm));
02428
02429 // Creating the variable widgets
02430 window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02431 gtk_widget_set_tooltip_text
02432     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02433 window->id_variable = g_signal_connect
02434     (window->combo_variable, "changed", window_set_variable, NULL);
02435 window->button_add_variable = (GtkButton *)
02436     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02437 g_signal_connect (window->button_add_variable, "clicked",
window_add_variable,
02438     NULL);
02439 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02440     _("Add variable"));
02441 window->button_remove_variable = (GtkButton *)
02442     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02443 g_signal_connect (window->button_remove_variable, "clicked",
02444     window_remove_variable, NULL);
02445 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02446     _("Remove variable"));
02447 window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02448 window->entry_variable = (GtkEntry *) gtk_entry_new ();
02449 gtk_widget_set_tooltip_text
02450     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02451 gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02452 window->id_variable_label = g_signal_connect
02453     (window->entry_variable, "changed", window_label_variable, NULL);
02454 window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02455 window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02456     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02457 gtk_widget_set_tooltip_text
02458     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02459 window->scrolled_min
02460     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02461 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02462     GTK_WIDGET (window->spin_min));
02463 g_signal_connect (window->spin_min, "value-changed",
02464     window_rangemin_variable, NULL);
02465 window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02466 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02467     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02468 gtk_widget_set_tooltip_text
02469     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02470 window->scrolled_max
02471     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02472 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02473     GTK_WIDGET (window->spin_max));
02474 g_signal_connect (window->spin_max, "value-changed",
02475     window_rangemax_variable, NULL);
02476 window->check_minabs = (GtkCheckButton *)
02477     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02478 g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02479 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02480     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02481 gtk_widget_set_tooltip_text
02482     (GTK_WIDGET (window->spin_minabs),
02483     _("Minimum allowed value of the variable"));
02484 window->scrolled_minabs
02485     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02486 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02487     GTK_WIDGET (window->spin_minabs));
02488 g_signal_connect (window->spin_minabs, "value-changed",
02489     window_rangeminabs_variable, NULL);
02490 window->check_maxabs = (GtkCheckButton *)
02491     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02492 g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02493 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02494     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02495 gtk_widget_set_tooltip_text
02496     (GTK_WIDGET (window->spin_maxabs),
02497     _("Maximum allowed value of the variable"));
02498 window->scrolled_maxabs
02499     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02500 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02501     GTK_WIDGET (window->spin_maxabs));
02502 g_signal_connect (window->spin_maxabs, "value-changed",
02503     window_rangemaxabs_variable, NULL);
02504 window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02505 window->spin_precision = (GtkSpinButton *)
02506     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02507 gtk_widget_set_tooltip_text
02508     (GTK_WIDGET (window->spin_precision),

```



```

02509     _("Number of precision floating point digits\n"
02510       "0 is for integer numbers"));
02511 g_signal_connect (window->spin_precision, "value-changed",
02512                  window_precision_variable, NULL);
02513 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02514 window->spin_sweeps =
02515   (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02516 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02517   _("Number of steps sweeping the variable"));
02518 g_signal_connect (window->spin_sweeps, "value-changed",
02519                  window_update_variable, NULL);
02520 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02521 window->spin_bits
02522   = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02523 gtk_widget_set_tooltip_text
02524   (GTK_WIDGET (window->spin_bits),
02525    _("Number of bits to encode the variable"));
02526 g_signal_connect
02527   (window->spin_bits, "value-changed", window_update_variable, NULL);
02528 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02529 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02530   (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02531 gtk_widget_set_tooltip_text
02532   (GTK_WIDGET (window->spin_step),
02533    _("Initial step size for the hill climbing method"));
02534 window->scrolled_step
02535   = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02536 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02537   GTK_WIDGET (window->spin_step));
02538 g_signal_connect
02539   (window->spin_step, "value-changed", window_step_variable, NULL);
02540 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02541 gtk_grid_attach (window->grid_variable,
02542   GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02543 gtk_grid_attach (window->grid_variable,
02544   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02545 gtk_grid_attach (window->grid_variable,
02546   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02547 gtk_grid_attach (window->grid_variable,
02548   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02549 gtk_grid_attach (window->grid_variable,
02550   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02551 gtk_grid_attach (window->grid_variable,
02552   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02553 gtk_grid_attach (window->grid_variable,
02554   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02555 gtk_grid_attach (window->grid_variable,
02556   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02557 gtk_grid_attach (window->grid_variable,
02558   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02559 gtk_grid_attach (window->grid_variable,
02560   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02561 gtk_grid_attach (window->grid_variable,
02562   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02563 gtk_grid_attach (window->grid_variable,
02564   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02565 gtk_grid_attach (window->grid_variable,
02566   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02567 gtk_grid_attach (window->grid_variable,
02568   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02569 gtk_grid_attach (window->grid_variable,
02570   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02571 gtk_grid_attach (window->grid_variable,
02572   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02573 gtk_grid_attach (window->grid_variable,
02574   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02575 gtk_grid_attach (window->grid_variable,
02576   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02577 gtk_grid_attach (window->grid_variable,
02578   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02579 gtk_grid_attach (window->grid_variable,
02580   GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02581 gtk_grid_attach (window->grid_variable,
02582   GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02583 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02584 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02585   GTK_WIDGET (window->grid_variable));
02586
02587 // Creating the experiment widgets
02588 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02589 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02590   _("Experiment selector"));
02591 window->id_experiment = g_signal_connect
02592   (window->combo_experiment, "changed", window_set_experiment, NULL);
02593
02594 window->button_add_experiment = (GtkButton *)
02595   gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);

```

```

02595 g_signal_connect
02596     (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02597 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02598     _("Add experiment"));
02599 window->button_remove_experiment = (GtkButton *)
02600     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02601 g_signal_connect (window->button_remove_experiment, "clicked",
02602     window_remove_experiment, NULL);
02603 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02604     _("Remove experiment"));
02605 window->label_experiment
02606     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02607 window->button_experiment = (GtkFileChooserButton *)
02608     gtk_file_chooser_button_new (_("Experimental data file"),
02609     GTK_FILE_CHOOSER_ACTION_OPEN);
02610 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02611     _("Experimental data file"));
02612 window->id_experiment_name
02613     = g_signal_connect (window->button_experiment, "selection-changed",
02614     window_name_experiment, NULL);
02615 gtk_widget_set_hexpend (GTK_WIDGET (window->button_experiment), TRUE);
02616 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02617 window->spin_weight
02618     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02619 gtk_widget_set_tooltip_text
02620     (GTK_WIDGET (window->spin_weight),
02621     _("Weight factor to build the objective function"));
02622 g_signal_connect
02623     (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
02624 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02625 gtk_grid_attach (window->grid_experiment,
02626     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02627 gtk_grid_attach (window->grid_experiment,
02628     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02629 gtk_grid_attach (window->grid_experiment,
02630     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02631 gtk_grid_attach (window->grid_experiment,
02632     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02633 gtk_grid_attach (window->grid_experiment,
02634     GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02635 gtk_grid_attach (window->grid_experiment,
02636     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02637 gtk_grid_attach (window->grid_experiment,
02638     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02639 for (i = 0; i < MAX_NINPUTS; ++i)
02640 {
02641     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02642     window->check_template[i] = (GtkCheckButton *)
02643         gtk_check_button_new_with_label (buffer3);
02644     window->id_template[i]
02645         = g_signal_connect (window->check_template[i], "toggled",
02646         window_inputs_experiment, NULL);
02647     gtk_grid_attach (window->grid_experiment,
02648         GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02649     window->button_template[i] = (GtkFileChooserButton *)
02650         gtk_file_chooser_button_new (_("Input template"),
02651         GTK_FILE_CHOOSER_ACTION_OPEN);
02652     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02653         _("Experimental input template file"));
02654     window->id_input[i] =
02655         g_signal_connect_swapped (window->button_template[i],
02656         "selection-changed",
02657         (GCallback) window_template_experiment,
02658         (void *) (size_t) i);
02659     gtk_grid_attach (window->grid_experiment,
02660         GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02661 }
02662 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02663 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02664     GTK_WIDGET (window->grid_experiment));
02665
02666 // Creating the error norm widgets
02667 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02668 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02669 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02670     GTK_WIDGET (window->grid_norm));
02671 window->button_norm[0] = (GtkRadioButton *)
02672     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02673 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02674     tip_norm[0]);
02675 gtk_grid_attach (window->grid_norm,
02676     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02677 g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02678 for (i = 0; ++i < NNORMS;)
02679 {

```

```

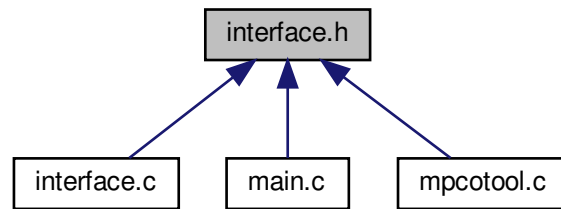
02680     window->button_norm[i] = (GtkRadioButton *)
02681         gtk_radio_button_new_with_mnemonic
02682         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02683     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02684         tip_norm[i]);
02685     gtk_grid_attach (window->grid_norm,
02686         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02687     g_signal_connect (window->button_norm[i], "clicked",
02688         window_update, NULL);
02689 }
02690 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02691 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02692 window->spin_p = (GtkSpinButton *)
02693     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02694 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02695     _("P parameter for the P error norm"));
02696 window->scrolled_p =
02697     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02698 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02699     GTK_WIDGET (window->spin_p));
02700 gtk_widget_set_hexpend (GTK_WIDGET (window->scrolled_p), TRUE);
02701 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02702 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02703     1, 2, 1, 2);
02704 // Creating the grid and attaching the widgets to the grid
02705 window->grid = (GtkGrid *) gtk_grid_new ();
02706 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02707 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02708 gtk_grid_attach (window->grid,
02709     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02710 gtk_grid_attach (window->grid,
02711     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02712 gtk_grid_attach (window->grid,
02713     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02714 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02715 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
02716     grid));
02717 // Setting the window logo
02718 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02719 gtk_window_set_icon (window->window, window->logo);
02720 // Showing the window
02721 gtk_widget_show_all (GTK_WIDGET (window->window));
02722 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02723 #if GTK_MINOR_VERSION >= 16
02724     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02725     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02726     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02727     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02728     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02729     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02730     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02731 #endif
02732 // Reading initial example
02733 input_new ();
02734 buffer2 = g_get_current_dir ();
02735 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02736 g_free (buffer2);
02737 window_read (buffer);
02738 g_free (buffer);
02739 #if DEBUG_INTERFACE
02740     fprintf (stderr, "window_new: start\n");
02741 #endif
02742 }

```

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

- unsigned int [gtk_array_get_active](#) (GtkRadioButton *array[], unsigned int n)
- void [input_save](#) (char *filename)
- void [options_new](#) ()
- void [running_new](#) ()
- unsigned int [window_get_algorithm](#) ()
- unsigned int [window_get_climbing](#) ()
- unsigned int [window_get_norm](#) ()
- void [window_save_climbing](#) ()
- int [window_save](#) ()
- void [window_run](#) ()
- void [window_help](#) ()
- void [window_update_climbing](#) ()
- void [window_update](#) ()
- void [window_set_algorithm](#) ()
- void [window_set_experiment](#) ()
- void [window_remove_experiment](#) ()
- void [window_add_experiment](#) ()
- void [window_name_experiment](#) ()
- void [window_weight_experiment](#) ()

- void [window_inputs_experiment](#) ()
- void [window_template_experiment](#) (void *data)
- void [window_set_variable](#) ()
- void [window_remove_variable](#) ()
- void [window_add_variable](#) ()
- void [window_label_variable](#) ()
- void [window_precision_variable](#) ()
- void [window_rangemin_variable](#) ()
- void [window_rangemax_variable](#) ()
- void [window_rangeminabs_variable](#) ()
- void [window_rangemaxabs_variable](#) ()
- void [window_update_variable](#) ()
- int [window_read](#) (char *filename)
- void [window_open](#) ()
- void [window_new](#) (GtkApplication *application)

Variables

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

4.13.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.13.2 Function Documentation

4.13.2.1 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

<i>array</i>	Array of GtkRadioButtons.
<i>n</i>	Number of GtkRadioButtons.

Definition at line 469 of file [utils.c](#).

```

00471 {
00472     unsigned int i;
00473     for (i = 0; i < n; ++i)
00474         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475             break;
00476     return i;
00477 }
```

4.13.2.2 input_save()

```

void input_save (
    char * filename )
```

Function to save the input file.

Parameters

<i>filename</i>	Input file name.
-----------------	----------------------------------

Definition at line 584 of file [interface.c](#).

```

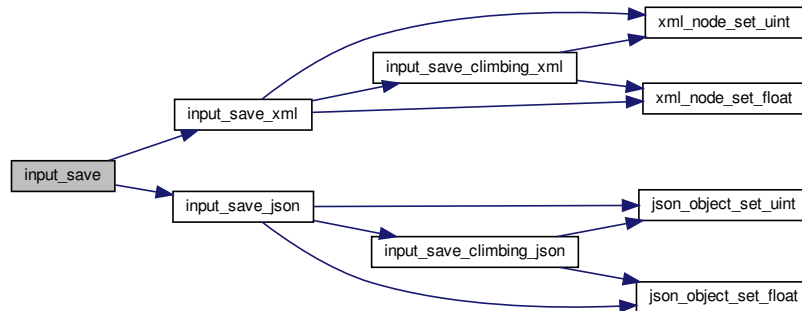
00585 {
00586     xmlDoc *doc;
00587     JsonGenerator *generator;
00588
00589     #if DEBUG_INTERFACE
00590     fprintf (stderr, "input_save: start\n");
00591     #endif
00592
00593     // Getting the input file directory
00594     input->name = g_path_get_basename (filename);
00595     input->directory = g_path_get_dirname (filename);
00596
00597     if (input->type == INPUT_TYPE_XML)
00598     {
00599         // Opening the input file
00600         doc = xmlNewDoc ((const xmlChar *) "1.0");
00601         input_save_xml (doc);
00602
00603         // Saving the XML file
00604         xmlSaveFormatFile (filename, doc, 1);
00605
00606         // Freeing memory
00607         xmlFreeDoc (doc);
00608     }
00609     else
00610     {
00611         // Opening the input file
00612         generator = json_generator_new ();
00613         json_generator_set_pretty (generator, TRUE);
00614         input_save_json (generator);
00615
00616         // Saving the JSON file
00617         json_generator_to_file (generator, filename, NULL);
00618
00619         // Freeing memory
```

```

00620     g_object_unref (generator);
00621 }
00622
00623 #if DEBUG_INTERFACE
00624 fprintf (stderr, "input_save: end\n");
00625 #endif
00626 }

```

Here is the call graph for this function:



4.13.2.3 options_new()

```
void options_new ( )
```

Function to open the options dialog.

Definition at line 632 of file [interface.c](#).

```

00633 {
00634 #if DEBUG_INTERFACE
00635 fprintf (stderr, "options_new: start\n");
00636 #endif
00637 options->label_seed = (GtkLabel *)
00638     gtk_label_new (_("Pseudo-random numbers generator seed"));
00639 options->spin_seed = (GtkSpinButton *)
00640     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00641 gtk_widget_set_tooltip_text
00642     (GTK_WIDGET (options->spin_seed),
00643      _("Seed to init the pseudo-random numbers generator"));
00644 gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00645 options->label_threads = (GtkLabel *)
00646     gtk_label_new (_("Threads number for the stochastic algorithm"));
00647 options->spin_threads
00648     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00649 gtk_widget_set_tooltip_text
00650     (GTK_WIDGET (options->spin_threads),
00651      _("Number of threads to perform the calibration/optimization for "
00652        "the stochastic algorithm"));
00653 gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00654 options->label_climbing = (GtkLabel *)
00655     gtk_label_new (_("Threads number for the hill climbing method"));
00656 options->spin_climbing =
00657     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00658 gtk_widget_set_tooltip_text
00659     (GTK_WIDGET (options->spin_climbing),
00660      _("Number of threads to perform the calibration/optimization for the "
00661        "hill climbing method"));
00662 gtk_spin_button_set_value (options->spin_climbing,

```

```

00663             (gdouble) nthreads_climbing);
00664     options->grid = (GtkGrid *) gtk_grid_new ();
00665     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_seed), 0, 0, 1, 1);
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_seed), 1, 0, 1, 1);
00667     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_threads),
00668                     0, 1, 1, 1);
00669     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_threads),
00670                     1, 1, 1, 1);
00671     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_climbing), 0, 2, 1,
00672                     1);
00673     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_climbing), 1, 2, 1,
00674                     1);
00675     gtk_widget_show_all (GTK_WIDGET (options->grid));
00676     options->dialog = (GtkDialog *)
00677         gtk_dialog_new_with_buttons (_("Options"),
00678                                     window->window,
00679                                     GTK_DIALOG_MODAL,
00680                                     _("_OK"), GTK_RESPONSE_OK,
00681                                     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00682     gtk_container_add
00683         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00684          GTK_WIDGET (options->grid));
00685     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00686     {
00687         input->seed
00688             = (unsigned long int) gtk_spin_button_get_value (options->
spin_seed);
00689         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00690         nthreads_climbing
00691             = gtk_spin_button_get_value_as_int (options->spin_climbing);
00692     }
00693     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00694     #if DEBUG_INTERFACE
00695     fprintf (stderr, "options_new: end\n");
00696     #endif
00697 }

```

4.13.2.4 running_new()

```
void running_new ( )
```

Function to open the running dialog.

Definition at line 703 of file [interface.c](#).

```

00704 {
00705     #if DEBUG_INTERFACE
00706     fprintf (stderr, "running_new: start\n");
00707     #endif
00708     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00709     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00710     running->grid = (GtkGrid *) gtk_grid_new ();
00711     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00712     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00713     running->dialog = (GtkDialog *)
00714         gtk_dialog_new_with_buttons (_("Calculating"),
00715                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00716     gtk_container_add (GTK_CONTAINER
00717         (gtk_dialog_get_content_area (running->dialog)),
00718         GTK_WIDGET (running->grid));
00719     gtk_spinner_start (running->spinner);
00720     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721     #if DEBUG_INTERFACE
00722     fprintf (stderr, "running_new: end\n");
00723     #endif
00724 }

```


4.13.2.5 window_add_experiment()

```
void window_add_experiment ( )
```

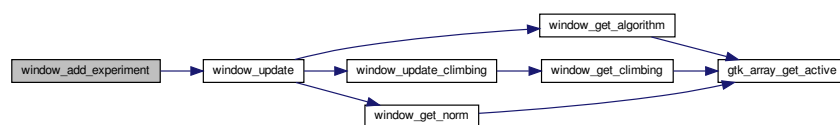
Function to add an experiment in the main window.

Definition at line 1392 of file [interface.c](#).

```

01393 {
01394     unsigned int i, j;
01395     #if DEBUG_INTERFACE
01396     fprintf (stderr, "window_add_experiment: start\n");
01397     #endif
01398     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01399     g_signal_handler_block (window->combo_experiment, window->
01400         id_experiment);
01401     gtk_combo_box_text_insert_text
01402         (window->combo_experiment, i, input->experiment[i].
01403         name);
01404     g_signal_handler_unblock (window->combo_experiment,
01405         window->id_experiment);
01406     input->experiment = (Experiment *) g_realloc
01407         (input->experiment, (input->nexperiments + 1) * sizeof (
01408         Experiment));
01409     for (j = input->nexperiments - 1; j > i; --j)
01410         mempcpy (input->experiment + j + 1, input->experiment + j,
01411             sizeof (Experiment));
01412     input->experiment[j + 1].weight = input->experiment[j].
01413     weight;
01414     input->experiment[j + 1].ninputs = input->
01415     experiment[j].ninputs;
01416     if (input->type == INPUT_TYPE_XML)
01417     {
01418         input->experiment[j + 1].name
01419             = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
01420             name);
01421         for (j = 0; j < input->experiment->ninputs; ++j)
01422             input->experiment[i + 1].stencil[j]
01423                 = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
01424                 stencil[j]);
01425     }
01426     else
01427     {
01428         input->experiment[j + 1].name = g_strdup (input->
01429         experiment[j].name);
01430         for (j = 0; j < input->experiment->ninputs; ++j)
01431             input->experiment[i + 1].stencil[j]
01432                 = g_strdup (input->experiment[i].stencil[j]);
01433     }
01434     ++input->nexperiments;
01435     for (j = 0; j < input->experiment->ninputs; ++j)
01436         g_signal_handler_block (window->button_template[j],
01437             window->id_input[j]);
01438     g_signal_handler_block
01439         (window->button_experiment, window->
01440         id_experiment_name);
01441     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01442     g_signal_handler_unblock
01443         (window->button_experiment, window->
01444         id_experiment_name);
01445     for (j = 0; j < input->experiment->ninputs; ++j)
01446         g_signal_handler_unblock (window->button_template[j],
01447             window->id_input[j]);
01448     window_update ();
01449     #if DEBUG_INTERFACE
01450     fprintf (stderr, "window_add_experiment: end\n");
01451     #endif
01452 }
```

Here is the call graph for this function:



4.13.2.6 window_add_variable()

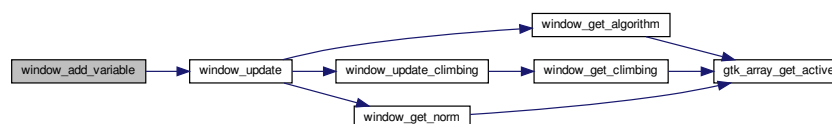
```
void window_add_variable ( )
```

Function to add a variable in the main window.

Definition at line 1655 of file [interface.c](#).

```
01656 {
01657     unsigned int i, j;
01658     #if DEBUG_INTERFACE
01659     fprintf (stderr, "window_add_variable: start\n");
01660     #endif
01661     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01662     g_signal_handler_block (window->combo_variable, window->
01663         id_variable);
01664     gtk_combo_box_text_insert_text (window->combo_variable, i,
01665         input->variable[i].name);
01666     g_signal_handler_unblock (window->combo_variable, window->
01667         id_variable);
01668     input->variable = (Variable *) g_realloc
01669         (input->variable, (input->nvariables + 1) * sizeof (
01670             Variable));
01671     for (j = input->nvariables - 1; j > i; --j)
01672         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01673             Variable));
01674     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01675             Variable));
01676     if (input->type == INPUT_TYPE_XML)
01677         input->variable[j + 1].name
01678             = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01679     else
01680         input->variable[j + 1].name = g_strdup (input->
01681             variable[j].name);
01682     ++input->nvariables;
01683     g_signal_handler_block (window->entry_variable, window->
01684         id_variable_label);
01685     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01686     g_signal_handler_unblock (window->entry_variable, window->
01687         id_variable_label);
01688     window_update ();
01689     #if DEBUG_INTERFACE
01690     fprintf (stderr, "window_add_variable: end\n");
01691     #endif
01692 }
```

Here is the call graph for this function:



4.13.2.7 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 732 of file [interface.c](#).

```
00733 {  
00734     unsigned int i;  
00735     #if DEBUG_INTERFACE  
00736     fprintf (stderr, "window_get_algorithm: start\n");  
00737     #endif  
00738     i = gtk_array_get_active (window->button_algorithm,  
        NATGORITHMS);  
00739     #if DEBUG_INTERFACE  
00740     fprintf (stderr, "window_get_algorithm: %u\n", i);  
00741     fprintf (stderr, "window_get_algorithm: end\n");  
00742     #endif  
00743     return i;  
00744 }
```

Here is the call graph for this function:



4.13.2.8 window_get_climbing()

```
unsigned int window_get_climbing ( )
```

Function to get the hill climbing method number.

Returns

Hill climbing method number.

Definition at line 752 of file [interface.c](#).

```
00753 {
00754     unsigned int i;
00755     #if DEBUG_INTERFACE
00756     fprintf (stderr, "window_get_climbing: start\n");
00757     #endif
00758     i = gtk_array_get_active (window->button_climbing,
00759                             NCLIMBINGS);
00759     #if DEBUG_INTERFACE
00760     fprintf (stderr, "window_get_climbing: %u\n", i);
00761     fprintf (stderr, "window_get_climbing: end\n");
00762     #endif
00763     return i;
00764 }
```

Here is the call graph for this function:



4.13.2.9 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

Returns

Norm method number.

Definition at line 772 of file [interface.c](#).

```
00773 {
00774     unsigned int i;
00775     #if DEBUG_INTERFACE
00776     fprintf (stderr, "window_get_norm: start\n");
00777     #endif
00778     i = gtk_array_get_active (window->button_norm,
00779                             NNORMS);
00779     #if DEBUG_INTERFACE
00780     fprintf (stderr, "window_get_norm: %u\n", i);
00781     fprintf (stderr, "window_get_norm: end\n");
00782     #endif
00783     return i;
00784 }
```

Here is the call graph for this function:



4.13.2.10 window_help()

```
void window_help ( )
```

Function to show a help dialog.

Definition at line 1029 of file [interface.c](#).

```

01030 {
01031     char *buffer, *buffer2;
01032     #if DEBUG_INTERFACE
01033     fprintf (stderr, "window_help: start\n");
01034     #endif
01035     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01036                               _("user-manual.pdf"), NULL);
01037     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01038     g_free (buffer2);
01039     #if GTK_MINOR_VERSION >= 22
01040     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01041     #else
01042     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01043     #endif
01044     #if DEBUG_INTERFACE
01045     fprintf (stderr, "window_help: uri=%s\n", buffer);
01046     #endif
01047     g_free (buffer);
01048     #if DEBUG_INTERFACE
01049     fprintf (stderr, "window_help: end\n");
01050     #endif
01051 }
```

4.13.2.11 window_inputs_experiment()

```
void window_inputs_experiment ( )
```

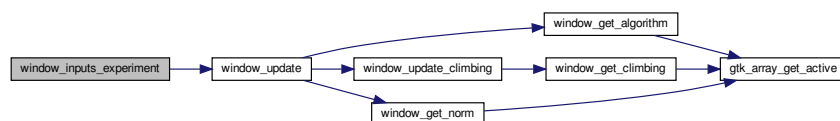
Function to update the experiment input templates number in the main window.

Definition at line 1492 of file [interface.c](#).

```

01493 {
01494     unsigned int j;
01495     #if DEBUG_INTERFACE
01496     fprintf (stderr, "window_inputs_experiment: start\n");
01497     #endif
01498     j = input->experiment->ninputs - 1;
01499     if (j
01500         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01501                                           (window->check_template[j])))
01502         --input->experiment->ninputs;
01503     if (input->experiment->ninputs < MAX_NINPUTS
01504         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01505                                           (window->check_template[j])))
01506         ++input->experiment->ninputs;
01507     window_update ();
01508     #if DEBUG_INTERFACE
01509     fprintf (stderr, "window_inputs_experiment: end\n");
01510     #endif
01511 }
```

Here is the call graph for this function:



4.13.2.12 window_label_variable()

```
void window_label_variable ( )
```

Function to set the variable label in the main window.

Definition at line 1690 of file [interface.c](#).

```
01691 {
01692     unsigned int i;
01693     const char *buffer;
01694     #if DEBUG_INTERFACE
01695     fprintf (stderr, "window_label_variable: start\n");
01696     #endif
01697     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01698     buffer = gtk_entry_get_text (window->entry_variable);
01699     g_signal_handler_block (window->combo_variable, window->
01700         id_variable);
01700     gtk_combo_box_text_remove (window->combo_variable, i);
01701     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01702     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01703     g_signal_handler_unblock (window->combo_variable, window->
01704         id_variable);
01704     #if DEBUG_INTERFACE
01705     fprintf (stderr, "window_label_variable: end\n");
01706     #endif
01707 }
```

4.13.2.13 window_name_experiment()

```
void window_name_experiment ( )
```

Function to set the experiment name in the main window.

Definition at line 1445 of file [interface.c](#).

```
01446 {
01447     unsigned int i;
01448     char *buffer;
01449     GFile *file1, *file2;
01450     #if DEBUG_INTERFACE
01451     fprintf (stderr, "window_name_experiment: start\n");
01452     #endif
01453     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01454     file1
01455     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
01456         button_experiment));
01456     file2 = g_file_new_for_path (input->directory);
01457     buffer = g_file_get_relative_path (file2, file1);
01458     g_signal_handler_block (window->combo_experiment, window->
01459         id_experiment);
01459     gtk_combo_box_text_remove (window->combo_experiment, i);
01460     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01461     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01462     g_signal_handler_unblock (window->combo_experiment,
01463         window->id_experiment);
01463     g_free (buffer);
01464     g_object_unref (file2);
01465     g_object_unref (file1);
01466     #if DEBUG_INTERFACE
01467     fprintf (stderr, "window_name_experiment: end\n");
01468     #endif
01469 }
```

4.13.2.14 window_new()

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

Function to open the main window.

Parameters

<i>application</i>	GtkApplication struct.
--------------------	------------------------

Definition at line 2065 of file [interface.c](#).

```

02066 {
02067     unsigned int i;
02068     char *buffer, *buffer2, buffer3[64];
02069     char *label_algorithm[NALGORITHMS] = {
02070         "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02071     };
02072     char *tip_algorithm[NALGORITHMS] = {
02073         _("Monte-Carlo brute force algorithm"),
02074         _("Sweep brute force algorithm"),
02075         _("Genetic algorithm"),
02076         _("Orthogonal sampling brute force algorithm"),
02077     };
02078     char *label_climbing[NCLIMBINGS] = {
02079         _("_Coordinates climbing"), _("_Random climbing")
02080     };
02081     char *tip_climbing[NCLIMBINGS] = {
02082         _("Coordinates climbing estimate method"),
02083         _("Random climbing estimate method")
02084     };
02085     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02086     char *tip_norm[NNORMS] = {
02087         _("Euclidean error norm (L2)"),
02088         _("Maximum error norm (L)"),
02089         _("P error norm (Lp)"),
02090         _("Taxicab error norm (L1)")
02091     };
02092
02093     #if DEBUG_INTERFACE
02094         fprintf(stderr, "window_new: start\n");
02095     #endif
02096
02097     // Creating the window
02098     window->window = main_window
02099         = (GtkWindow *) gtk_application_window_new (application);
02100
02101     // Finish when closing the window
02102     g_signal_connect_swapped (window->window, "delete-event",
02103                             G_CALLBACK (g_application_quit),
02104                             G_APPLICATION (application));
02105
02106     // Setting the window title
02107     gtk_window_set_title (window->window, "MPCOTool");
02108
02109     // Creating the open button
02110     window->button_open = (GtkToolButton *) gtk_tool_button_new
02111         (gtk_image_new_from_icon_name ("document-open",
02112                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02113     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02114
02115     // Creating the save button
02116     window->button_save = (GtkToolButton *) gtk_tool_button_new
02117         (gtk_image_new_from_icon_name ("document-save",
02118                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02119     g_signal_connect (window->button_save, "clicked", (GCallback)
02120 window_save,
02121                     NULL);
02122
02123     // Creating the run button
02124     window->button_run = (GtkToolButton *) gtk_tool_button_new
02125         (gtk_image_new_from_icon_name ("system-run",
02126                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02127     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02128
02129     // Creating the options button
02130     window->button_options = (GtkToolButton *) gtk_tool_button_new
02131         (gtk_image_new_from_icon_name ("preferences-system",
02132                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02133     g_signal_connect (window->button_options, "clicked",
02134 options_new, NULL);
02135
02136     // Creating the help button
02137     window->button_help = (GtkToolButton *) gtk_tool_button_new
02138         (gtk_image_new_from_icon_name ("help-browser",
02139                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02140     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02141

```

```

02140 // Creating the about button
02141 window->button_about = (GtkToolButton *) gtk_tool_button_new
02142     (gtk_image_new_from_icon_name ("help-about",
02143         GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02144 g_signal_connect (window->button_about, "clicked",
02145     window_about, NULL);
02146 // Creating the exit button
02147 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02148     (gtk_image_new_from_icon_name ("application-exit",
02149         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02150 g_signal_connect_swapped (window->button_exit, "clicked",
02151     G_CALLBACK (g_application_quit),
02152     G_APPLICATION (application));
02153
02154 // Creating the buttons bar
02155 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02156 gtk_toolbar_insert
02157     (window->bar_buttons, GTK_TOOL_ITEM (window->
02158     button_open), 0);
02159 gtk_toolbar_insert
02160     (window->bar_buttons, GTK_TOOL_ITEM (window->
02161     button_save), 1);
02162 gtk_toolbar_insert
02163     (window->bar_buttons, GTK_TOOL_ITEM (window->
02164     button_run), 2);
02165 gtk_toolbar_insert
02166     (window->bar_buttons, GTK_TOOL_ITEM (window->
02167     button_options), 3);
02168 gtk_toolbar_insert
02169     (window->bar_buttons, GTK_TOOL_ITEM (window->
02170     button_help), 4);
02171 gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->
02173     button_about), 5);
02174 gtk_toolbar_insert
02175     (window->bar_buttons, GTK_TOOL_ITEM (window->
02176     button_exit), 6);
02177 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02178
02179 // Creating the simulator program label and entry
02180 window->label_simulator = (GtkLabel *) gtk_label_new _("Simulator program");
02181 window->button_simulator = (GtkFileChooserButton *)
02182     gtk_file_chooser_button_new _("Simulator program"),
02183     GTK_FILE_CHOOSER_ACTION_OPEN);
02184 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02185     _("Simulator program executable file"));
02186 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02187
02188 // Creating the evaluator program label and entry
02189 window->check_evaluator = (GtkCheckButton *)
02190     gtk_check_button_new_with_mnemonic _("Evaluator program");
02191 g_signal_connect (window->check_evaluator, "toggled",
02192     window_update, NULL);
02193 window->button_evaluator = (GtkFileChooserButton *)
02194     gtk_file_chooser_button_new _("Evaluator program"),
02195     GTK_FILE_CHOOSER_ACTION_OPEN);
02196 gtk_widget_set_tooltip_text
02197     (GTK_WIDGET (window->button_evaluator),
02198     _("Optional evaluator program executable file"));
02199
02200 // Creating the results files labels and entries
02201 window->label_result = (GtkLabel *) gtk_label_new _("Result file");
02202 window->entry_result = (GtkEntry *) gtk_entry_new ();
02203 gtk_widget_set_tooltip_text
02204     (GTK_WIDGET (window->entry_result), _("Best results file"));
02205 window->label_variables = (GtkLabel *) gtk_label_new _("Variables file");
02206 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02207 gtk_widget_set_tooltip_text
02208     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02209
02210 // Creating the files grid and attaching widgets
02211 window->grid_files = (GtkGrid *) gtk_grid_new ();
02212 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02213     label_simulator),
02214     0, 0, 1, 1);
02215 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02216     button_simulator),
02217     1, 0, 1, 1);
02218 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02219     check_evaluator),
02220     0, 1, 1, 1);
02221 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02222     button_evaluator),
02223     1, 1, 1, 1);
02224 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02225     label_result),

```



```

02213         0, 2, 1, 1);
02214     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02215         1, 2, 1, 1);
02216     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02217         0, 3, 1, 1);
02218     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02219         1, 3, 1, 1);
02220
02221     // Creating the algorithm properties
02222     window->label_simulations = (GtkLabel *) gtk_label_new
02223     (_("Simulations number"));
02224     window->spin_simulations
02225     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02226     gtk_widget_set_tooltip_text
02227     (GTK_WIDGET (window->spin_simulations),
02228     _("Number of simulations to perform for each iteration"));
02229     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02230     window->label_iterations = (GtkLabel *)
02231     gtk_label_new (_("Iterations number"));
02232     window->spin_iterations
02233     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02234     gtk_widget_set_tooltip_text
02235     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02236     g_signal_connect
02237     (window->spin_iterations, "value-changed",
window_update, NULL);
02238     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02239     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02240     window->spin_tolerance =
02241     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02242     gtk_widget_set_tooltip_text
02243     (GTK_WIDGET (window->spin_tolerance),
02244     _("Tolerance to set the variable interval on the next iteration"));
02245     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02246     window->spin_bests
02247     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02248     gtk_widget_set_tooltip_text
02249     (GTK_WIDGET (window->spin_bests),
02250     _("Number of best simulations used to set the variable interval "
02251     "on the next iteration"));
02252     window->label_population
02253     = (GtkLabel *) gtk_label_new (_("Population number"));
02254     window->spin_population
02255     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02256     gtk_widget_set_tooltip_text
02257     (GTK_WIDGET (window->spin_population),
02258     _("Number of population for the genetic algorithm"));
02259     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02260     window->label_generations
02261     = (GtkLabel *) gtk_label_new (_("Generations number"));
02262     window->spin_generations
02263     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02264     gtk_widget_set_tooltip_text
02265     (GTK_WIDGET (window->spin_generations),
02266     _("Number of generations for the genetic algorithm"));
02267     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02268     window->spin_mutation
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02270     gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_mutation),
02272     _("Ratio of mutation for the genetic algorithm"));
02273     window->label_reproduction
02274     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02275     window->spin_reproduction
02276     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02277     gtk_widget_set_tooltip_text
02278     (GTK_WIDGET (window->spin_reproduction),
02279     _("Ratio of reproduction for the genetic algorithm"));
02280     window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02281     window->spin_adaptation
02282     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02283     gtk_widget_set_tooltip_text
02284     (GTK_WIDGET (window->spin_adaptation),
02285     _("Ratio of adaptation for the genetic algorithm"));
02286     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02287     window->spin_threshold = (GtkSpinButton *)
02288     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02289     precision[DEFAULT_PRECISION]);
02290     gtk_widget_set_tooltip_text
02291     (GTK_WIDGET (window->spin_threshold),
02292     _("Threshold in the objective function to finish the simulations"));
02293     window->scrolled_threshold =
02294     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02295     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),

```

```

02296             GTK_WIDGET (window->spin_threshold));
02297 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02298 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02299 //             GTK_ALIGN_FILL);
02300
02301 // Creating the hill climbing method properties
02302 window->check_climbing = (GtkCheckButton *)
02303     gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02304 g_signal_connect (window->check_climbing, "clicked",
window_update, NULL);
02305 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02306 window->button_climbing[0] = (GtkRadioButton *)
02307     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02308 gtk_grid_attach (window->grid_climbing,
02309     GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02310 g_signal_connect (window->button_climbing[0], "clicked",
window_update, NULL);
02311 for (i = 0; ++i < NCLIMBINGS;)
02312 {
02313     window->button_climbing[i] = (GtkRadioButton *)
02314         gtk_radio_button_new_with_mnemonic
02315             (gtk_radio_button_get_group (window->button_climbing[0]),
02316             label_climbing[i]);
02317     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02318         tip_climbing[i]);
02319     gtk_grid_attach (window->grid_climbing,
02320         GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02321     g_signal_connect (window->button_climbing[i], "clicked",
window_update,
02322         NULL);
02323 }
02324 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02325 window->spin_steps = (GtkSpinButton *)
02326     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02327 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02328 window->label_estimates
02329     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02330 window->spin_estimates = (GtkSpinButton *)
02331     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02332 window->label_relaxation
02333     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02334 window->spin_relaxation = (GtkSpinButton *)
02335     gtk_spin_button_new_with_range (0., 2., 0.001);
02336 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_steps),
02337     0, NCLIMBINGS, 1, 1);
02338 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_steps),
02339     1, NCLIMBINGS, 1, 1);
02340 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_estimates),
02341     0, NCLIMBINGS + 1, 1, 1);
02342 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_estimates),
02343     1, NCLIMBINGS + 1, 1, 1);
02344 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_relaxation),
02345     0, NCLIMBINGS + 2, 1, 1);
02346 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_relaxation),
02347     1, NCLIMBINGS + 2, 1, 1);
02348
02349 // Creating the array of algorithms
02350 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02351 window->button_algorithm[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02353 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02354     tip_algorithm[0]);
02355 gtk_grid_attach (window->grid_algorithm,
02356     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02357 g_signal_connect (window->button_algorithm[0], "clicked",
02358     window_set_algorithm, NULL);
02359 for (i = 0; ++i < NALGORITHMS;)
02360 {
02361     window->button_algorithm[i] = (GtkRadioButton *)
02362         gtk_radio_button_new_with_mnemonic
02363             (gtk_radio_button_get_group (window->button_algorithm[0]),
02364             label_algorithm[i]);
02365     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02366         tip_algorithm[i]);
02367     gtk_grid_attach (window->grid_algorithm,
02368         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02369     g_signal_connect (window->button_algorithm[i], "clicked",
02370         window_set_algorithm, NULL);
02371 }
02372 gtk_grid_attach (window->grid_algorithm,
02373     GTK_WIDGET (window->label_simulations),

```

```

02374         0, NALGORITHMS, 1, 1);
02375     gtk_grid_attach (window->grid_algorithm,
02376         GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02377     gtk_grid_attach (window->grid_algorithm,
02378         GTK_WIDGET (window->label_iterations),
02379         0, NALGORITHMS + 1, 1, 1);
02380     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02381         window->spin_iterations),
02382         1, NALGORITHMS + 1, 1, 1);
02383     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02384         window->label_tolerance),
02385         0, NALGORITHMS + 2, 1, 1);
02386     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02387         window->spin_tolerance),
02388         1, NALGORITHMS + 2, 1, 1);
02389     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02390         window->label_bests),
02391         0, NALGORITHMS + 3, 1, 1);
02392     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02393         window->spin_bests),
02394         1, NALGORITHMS + 3, 1, 1);
02395     gtk_grid_attach (window->grid_algorithm,
02396         GTK_WIDGET (window->label_population),
02397         0, NALGORITHMS + 4, 1, 1);
02398     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02399         window->spin_population),
02400         1, NALGORITHMS + 4, 1, 1);
02401     gtk_grid_attach (window->grid_algorithm,
02402         GTK_WIDGET (window->label_generations),
02403         0, NALGORITHMS + 5, 1, 1);
02404     gtk_grid_attach (window->grid_algorithm,
02405         GTK_WIDGET (window->spin_generations),
02406         1, NALGORITHMS + 5, 1, 1);
02407     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02408         window->label_mutation),
02409         0, NALGORITHMS + 6, 1, 1);
02410     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02411         window->spin_mutation),
02412         1, NALGORITHMS + 6, 1, 1);
02413     gtk_grid_attach (window->grid_algorithm,
02414         GTK_WIDGET (window->label_reproduction),
02415         0, NALGORITHMS + 7, 1, 1);
02416     gtk_grid_attach (window->grid_algorithm,
02417         GTK_WIDGET (window->spin_reproduction),
02418         1, NALGORITHMS + 7, 1, 1);
02419     gtk_grid_attach (window->grid_algorithm,
02420         GTK_WIDGET (window->label_adaptation),
02421         0, NALGORITHMS + 8, 1, 1);
02422     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02423         window->spin_adaptation),
02424         1, NALGORITHMS + 8, 1, 1);
02425     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02426         window->check_climbing),
02427         0, NALGORITHMS + 9, 2, 1);
02428     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02429         window->grid_climbing),
02430         0, NALGORITHMS + 10, 2, 1);
02431     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02432         window->label_threshold),
02433         0, NALGORITHMS + 11, 1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435         GTK_WIDGET (window->scrolled_threshold),
02436         1, NALGORITHMS + 11, 1, 1);
02437     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02438     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02439         GTK_WIDGET (window->grid_algorithm));
02440
02441     // Creating the variable widgets
02442     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02443     gtk_widget_set_tooltip_text
02444         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02445     window->id_variable = g_signal_connect
02446         (window->combo_variable, "changed", window_set_variable, NULL);
02447     window->button_add_variable = (GtkButton *)
02448         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02449     g_signal_connect (window->button_add_variable, "clicked",
02450         window_add_variable,
02451         NULL);
02452     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02453         _("Add variable"));
02454     window->button_remove_variable = (GtkButton *)
02455         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02456     g_signal_connect (window->button_remove_variable, "clicked",
02457         window_remove_variable, NULL);
02458     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02459         _("Remove variable"));
02460     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));

```

```

02448     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02449     gtk_widget_set_tooltip_text
02450         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02451     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02452     window->id_variable_label = g_signal_connect
02453         (window->entry_variable, "changed",
window_label_variable, NULL);
02454     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02455     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02456         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02457     gtk_widget_set_tooltip_text
02458         (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02459     window->scrolled_min
02460         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02461     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02462         GTK_WIDGET (window->spin_min));
02463     g_signal_connect (window->spin_min, "value-changed",
02464         window_rangemin_variable, NULL);
02465     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02466     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02467         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02468     gtk_widget_set_tooltip_text
02469         (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02470     window->scrolled_max
02471         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02472     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02473         GTK_WIDGET (window->spin_max));
02474     g_signal_connect (window->spin_max, "value-changed",
02475         window_rangemax_variable, NULL);
02476     window->check_minabs = (GtkCheckButton *)
02477         gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02478     g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
02479     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02480         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02481     gtk_widget_set_tooltip_text
02482         (GTK_WIDGET (window->spin_minabs),
02483         _("Minimum allowed value of the variable"));
02484     window->scrolled_minabs
02485         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02486     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02487         GTK_WIDGET (window->spin_minabs));
02488     g_signal_connect (window->spin_minabs, "value-changed",
02489         window_rangeminabs_variable, NULL);
02490     window->check_maxabs = (GtkCheckButton *)
02491         gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02492     g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02493     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02494         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02495     gtk_widget_set_tooltip_text
02496         (GTK_WIDGET (window->spin_maxabs),
02497         _("Maximum allowed value of the variable"));
02498     window->scrolled_maxabs
02499         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02500     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02501         GTK_WIDGET (window->spin_maxabs));
02502     g_signal_connect (window->spin_maxabs, "value-changed",
02503         window_rangemaxabs_variable, NULL);
02504     window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02505     window->spin_precision = (GtkSpinButton *)
02506         gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02507     gtk_widget_set_tooltip_text
02508         (GTK_WIDGET (window->spin_precision),
02509         _("Number of precision floating point digits\n"
02510         "0 is for integer numbers"));
02511     g_signal_connect (window->spin_precision, "value-changed",
02512         window_precision_variable, NULL);
02513     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02514     window->spin_sweeps =
02515         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02516     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02517         _("Number of steps sweeping the variable"));
02518     g_signal_connect (window->spin_sweeps, "value-changed",
02519         window_update_variable, NULL);
02520     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02521     window->spin_bits
02522         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02523     gtk_widget_set_tooltip_text
02524         (GTK_WIDGET (window->spin_bits),
02525         _("Number of bits to encode the variable"));
02526     g_signal_connect
02527         (window->spin_bits, "value-changed", window_update_variable, NULL);
;
02528     window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02529     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02530         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);

```

```

02531 gtk_widget_set_tooltip_text
02532     (GTK_WIDGET (window->spin_step),
02533      _("Initial step size for the hill climbing method"));
02534 window->scrolled_step
02535     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02536 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02537                    GTK_WIDGET (window->spin_step));
02538 g_signal_connect
02539     (window->spin_step, "value-changed", window_step_variable, NULL);
02540 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02541 gtk_grid_attach (window->grid_variable,
02542                 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02543 gtk_grid_attach (window->grid_variable,
02544                 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02545 gtk_grid_attach (window->grid_variable,
02546                 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02547 gtk_grid_attach (window->grid_variable,
02548                 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02549 gtk_grid_attach (window->grid_variable,
02550                 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02551 gtk_grid_attach (window->grid_variable,
02552                 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02553 gtk_grid_attach (window->grid_variable,
02554                 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02555 gtk_grid_attach (window->grid_variable,
02556                 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02557 gtk_grid_attach (window->grid_variable,
02558                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02559 gtk_grid_attach (window->grid_variable,
02560                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02561 gtk_grid_attach (window->grid_variable,
02562                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02563 gtk_grid_attach (window->grid_variable,
02564                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02565 gtk_grid_attach (window->grid_variable,
02566                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02567 gtk_grid_attach (window->grid_variable,
02568                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02569 gtk_grid_attach (window->grid_variable,
02570                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02571 gtk_grid_attach (window->grid_variable,
02572                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02573 gtk_grid_attach (window->grid_variable,
02574                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02575 gtk_grid_attach (window->grid_variable,
02576                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02577 gtk_grid_attach (window->grid_variable,
02578                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02579 gtk_grid_attach (window->grid_variable,
02580                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02581 gtk_grid_attach (window->grid_variable,
02582                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02583 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02584 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02585                    GTK_WIDGET (window->grid_variable));
02586
02587 // Creating the experiment widgets
02588 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02589 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02590                              _("Experiment selector"));
02591 window->id_experiment = g_signal_connect
02592     (window->combo_experiment, "changed",
02593      window_set_experiment, NULL);
02594 window->button_add_experiment = (GtkButton *)
02595     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02596 g_signal_connect
02597     (window->button_add_experiment, "clicked",
02598      window_add_experiment, NULL);
02599 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02600                              _("Add experiment"));
02601 window->button_remove_experiment = (GtkButton *)
02602     gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02603 g_signal_connect (window->button_remove_experiment, "clicked",
02604                  window_remove_experiment, NULL);
02605 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
02606 button_remove_experiment),
02607                              _("Remove experiment"));
02608 window->label_experiment
02609     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02610 window->button_experiment = (GtkFileChooserButton *)
02611     gtk_file_chooser_button_new (_("Experimental data file"),
02612                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02613 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02614                              _("Experimental data file"));
02615 window->id_experiment_name
02616     = g_signal_connect (window->button_experiment, "selection-changed",
02617                        window_name_experiment, NULL);

```

```

02615 gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02616 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02617 window->spin_weight
02618 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02619 gtk_widget_set_tooltip_text
02620 (GTK_WIDGET (window->spin_weight),
02621 _("Weight factor to build the objective function"));
02622 g_signal_connect
02623 (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
02624 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02625 gtk_grid_attach (window->grid_experiment,
02626 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02627 gtk_grid_attach (window->grid_experiment,
02628 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02629 gtk_grid_attach (window->grid_experiment,
02630 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
;
02631 gtk_grid_attach (window->grid_experiment,
02632 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02633 gtk_grid_attach (window->grid_experiment,
02634 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02635 gtk_grid_attach (window->grid_experiment,
02636 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02637 gtk_grid_attach (window->grid_experiment,
02638 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02639 for (i = 0; i < MAX_NINPITS; ++i)
02640 {
02641     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02642     window->check_template[i] = (GtkCheckButton *)
02643     gtk_check_button_new_with_label (buffer3);
02644     window->id_template[i]
02645     = g_signal_connect (window->check_template[i], "toggled",
02646     window_inputs_experiment, NULL);
02647     gtk_grid_attach (window->grid_experiment,
02648     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02649     window->button_template[i] = (GtkFileChooserButton *)
02650     gtk_file_chooser_button_new (_("Input template"),
02651     GTK_FILE_CHOOSER_ACTION_OPEN);
02652     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02653     _("Experimental input template file"));
02654     window->id_input[i] =
02655     g_signal_connect_swapped (window->button_template[i],
02656     "selection-changed",
02657     (GCallback) window_template_experiment,
02658     (void *) (size_t) i);
02659     gtk_grid_attach (window->grid_experiment,
02660     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02661 }
02662 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02663 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02664 GTK_WIDGET (window->grid_experiment));
02665
02666 // Creating the error norm widgets
02667 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02668 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02669 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02670 GTK_WIDGET (window->grid_norm));
02671 window->button_norm[0] = (GtkRadioButton *)
02672     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02673 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02674     tip_norm[0]);
02675 gtk_grid_attach (window->grid_norm,
02676     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02677 g_signal_connect (window->button_norm[0], "clicked",
window_update, NULL);
02678 for (i = 0; ++i < NNORMS;)
02679 {
02680     window->button_norm[i] = (GtkRadioButton *)
02681     gtk_radio_button_new_with_mnemonic
02682     (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02683     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02684     tip_norm[i]);
02685     gtk_grid_attach (window->grid_norm,
02686     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02687     g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
02688 }
02689 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02690 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
label_p), 1, 1, 1, 1);
02691 window->spin_p = (GtkSpinButton *)
02692     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02693 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02694     _("P parameter for the P error norm"));
02695 window->scrolled_p =
02696     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);

```

```

02697     gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02698                        GTK_WIDGET (window->spin_p));
02699     gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02700     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02701     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
scrolled_p),
02702                     1, 2, 1, 2);
02703
02704     // Creating the grid and attaching the widgets to the grid
02705     window->grid = (GtkGrid *) gtk_grid_new ();
02706     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02707     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02708     gtk_grid_attach (window->grid,
02709                     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02710     gtk_grid_attach (window->grid,
02711                     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02712     gtk_grid_attach (window->grid,
02713                     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02714     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02715     gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
window->grid));
02716
02717     // Setting the window logo
02718     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02719     gtk_window_set_icon (window->window, window->logo);
02720
02721     // Showing the window
02722     gtk_widget_show_all (GTK_WIDGET (window->window));
02723
02724     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02725     #if GTK_MINOR_VERSION >= 16
02726     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02727     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02728     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02729     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02730     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02731     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02732     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02733     #endif
02734
02735     // Reading initial example
02736     input_new ();
02737     buffer2 = g_get_current_dir ();
02738     buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02739     g_free (buffer2);
02740     window_read (buffer);
02741     g_free (buffer);
02742
02743     #if DEBUG_INTERFACE
02744     fprintf (stderr, "window_new: start\n");
02745     #endif
02746 }

```

4.13.2.15 window_open()

```
void window_open ( )
```

Function to open the input data.

Definition at line 1979 of file [interface.c](#).

```

01980 {
01981     GtkFileChooserDialog *dlg;
01982     GtkFileFilter *filter;
01983     char *buffer, *directory, *name;
01984
01985     #if DEBUG_INTERFACE
01986     fprintf (stderr, "window_open: start\n");
01987     #endif
01988
01989     // Saving a backup of the current input file
01990     directory = g_strdup (input->directory);
01991     name = g_strdup (input->name);
01992
01993     // Opening dialog

```



```

01994     dlg = (GtkFileChooserDialog *)
01995         gtk_file_chooser_dialog_new (_("Open input file"),
01996                                     window->window,
01997                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01998                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
01999                                     _("_OK"), GTK_RESPONSE_OK, NULL);
02000
02001     // Adding XML filter
02002     filter = (GtkFileFilter *) gtk_file_filter_new ();
02003     gtk_file_filter_set_name (filter, "XML");
02004     gtk_file_filter_add_pattern (filter, "*.xml");
02005     gtk_file_filter_add_pattern (filter, "*.XML");
02006     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02007
02008     // Adding JSON filter
02009     filter = (GtkFileFilter *) gtk_file_filter_new ();
02010     gtk_file_filter_set_name (filter, "JSON");
02011     gtk_file_filter_add_pattern (filter, "*.json");
02012     gtk_file_filter_add_pattern (filter, "*.JSON");
02013     gtk_file_filter_add_pattern (filter, "*.js");
02014     gtk_file_filter_add_pattern (filter, "*.JS");
02015     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02016
02017     // If OK saving
02018     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02019     {
02020
02021         // Traying to open the input file
02022         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02023         if (!window_read (buffer))
02024         {
02025             #if DEBUG_INTERFACE
02026                 fprintf (stderr, "window_open: error reading input file\n");
02027             #endif
02028             g_free (buffer);
02029
02030             // Reading backup file on error
02031             buffer = g_build_filename (directory, name, NULL);
02032             input->result = input->variables = NULL;
02033             if (!input_open (buffer))
02034             {
02035
02036                 // Closing on backup file reading error
02037                 #if DEBUG_INTERFACE
02038                     fprintf (stderr, "window_read: error reading backup file\n");
02039                 #endif
02040                 g_free (buffer);
02041                 break;
02042             }
02043             g_free (buffer);
02044         }
02045         else
02046         {
02047             g_free (buffer);
02048             break;
02049         }
02050     }
02051
02052     // Freeing and closing
02053     g_free (name);
02054     g_free (directory);
02055     gtk_widget_destroy (GTK_WIDGET (dlg));
02056     #if DEBUG_INTERFACE
02057         fprintf (stderr, "window_open: end\n");
02058     #endif
02059 }

```

4.13.2.16 window_precision_variable()

```
void window_precision_variable ( )
```

Function to update the variable precision in the main window.

Definition at line 1713 of file [interface.c](#).


```

01714 {
01715     unsigned int i;
01716     #if DEBUG_INTERFACE
01717         fprintf (stderr, "window_precision_variable: start\n");
01718     #endif
01719     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01720     input->variable[i].precision
01721     = (unsigned int) gtk_spin_button_get_value_as_int (window->
    spin_precision);
01722     gtk_spin_button_set_digits (window->spin_min, input->
    variable[i].precision);
01723     gtk_spin_button_set_digits (window->spin_max, input->
    variable[i].precision);
01724     gtk_spin_button_set_digits (window->spin_minabs,
    input->variable[i].precision);
01725     gtk_spin_button_set_digits (window->spin_maxabs,
    input->variable[i].precision);
01726     #if DEBUG_INTERFACE
01727         fprintf (stderr, "window_precision_variable: end\n");
01728     #endif
01729 }

```

4.13.2.17 window_rangemax_variable()

```
void window_rangemax_variable ( )
```

Function to update the variable rangemax in the main window.

Definition at line 1754 of file [interface.c](#).

```

01755 {
01756     unsigned int i;
01757     #if DEBUG_INTERFACE
01758         fprintf (stderr, "window_rangemax_variable: start\n");
01759     #endif
01760     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01761     input->variable[i].rangemax = gtk_spin_button_get_value (
    window->spin_max);
01762     #if DEBUG_INTERFACE
01763         fprintf (stderr, "window_rangemax_variable: end\n");
01764     #endif
01765 }

```

4.13.2.18 window_rangemaxabs_variable()

```
void window_rangemaxabs_variable ( )
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1789 of file [interface.c](#).

```

01790 {
01791     unsigned int i;
01792     #if DEBUG_INTERFACE
01793         fprintf (stderr, "window_rangemaxabs_variable: start\n");
01794     #endif
01795     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01796     input->variable[i].rangemaxabs
01797     = gtk_spin_button_get_value (window->spin_maxabs);
01798     #if DEBUG_INTERFACE
01799         fprintf (stderr, "window_rangemaxabs_variable: end\n");
01800     #endif
01801 }

```

4.13.2.19 window_rangemin_variable()

```
void window_rangemin_variable ( )
```

Function to update the variable rangemin in the main window.

Definition at line 1737 of file [interface.c](#).

```
01738 {
01739     unsigned int i;
01740     #if DEBUG_INTERFACE
01741         fprintf (stderr, "window_rangemin_variable: start\n");
01742     #endif
01743     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01744     input->variable[i].rangemin = gtk_spin_button_get_value (
    window->spin_min);
01745     #if DEBUG_INTERFACE
01746         fprintf (stderr, "window_rangemin_variable: end\n");
01747     #endif
01748 }
```

4.13.2.20 window_rangeminabs_variable()

```
void window_rangeminabs_variable ( )
```

Function to update the variable rangeminabs in the main window.

Definition at line 1771 of file [interface.c](#).

```
01772 {
01773     unsigned int i;
01774     #if DEBUG_INTERFACE
01775         fprintf (stderr, "window_rangeminabs_variable: start\n");
01776     #endif
01777     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01778     input->variable[i].rangeminabs
    = gtk_spin_button_get_value (window->spin_minabs);
01780     #if DEBUG_INTERFACE
01781         fprintf (stderr, "window_rangeminabs_variable: end\n");
01782     #endif
01783 }
```

4.13.2.21 window_read()

```
int window_read (
    char * filename )
```

Function to read the input data of a file.

Returns

1 on succes, 0 on error.

Parameters

<i>filename</i>	File name.
-----------------	------------

Definition at line 1863 of file [interface.c](#).

```

01864 {
01865     unsigned int i;
01866     char *buffer;
01867     #if DEBUG_INTERFACE
01868     fprintf (stderr, "window_read: start\n");
01869     #endif
01870
01871     // Reading new input file
01872     input_free ();
01873     input->result = input->variables = NULL;
01874     if (!input_open (filename))
01875     {
01876     #if DEBUG_INTERFACE
01877         fprintf (stderr, "window_read: end\n");
01878     #endif
01879         return 0;
01880     }
01881
01882     // Setting GTK+ widgets data
01883     gtk_entry_set_text (window->entry_result, input->result);
01884     gtk_entry_set_text (window->entry_variables, input->
variables);
01885     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01886     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01887     g_free (buffer);
01888     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01889     if (input->evaluator)
01890     {
01891         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01892         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01893         g_free (buffer);
01894     }
01895     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01896     switch (input->algorithm)
01897     {
01898     case ALGORITHM_MONTE_CARLO:
01899         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01900         // fallthrough
01901     case ALGORITHM_SWEEP:
01902     case ALGORITHM_ORTHOGONAL:
01903         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01904         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01905         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01906         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_climbing),
input->nsteps);
01907         if (input->nsteps)
01908         {
01909             gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_climbing[
input->climbing]),
TRUE);
01910             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01911             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01912             switch (input->climbing)
01913             {
01914             case CLIMBING_METHOD_RANDOM:
01915                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01916             }
01917         }
01918         break;
01919     default:

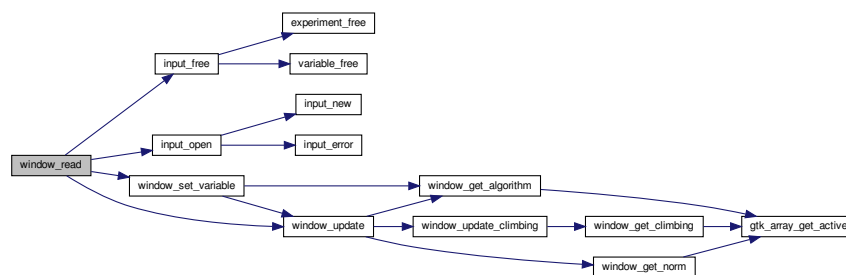
```

```

01932     gtk_spin_button_set_value (window->spin_population,
01933                               (gdouble) input->nsimulations);
01934     gtk_spin_button_set_value (window->spin_generations,
01935                               (gdouble) input->niterations);
01936     gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01937     gtk_spin_button_set_value (window->spin_reproduction,
01938                               input->reproduction_ratio);
01939     gtk_spin_button_set_value (window->spin_adaptation,
01940                               input->adaptation_ratio);
01941 }
01942 gtk_toggle_button_set_active
01943 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01944 gtk_spin_button_set_value (window->spin_p, input->p);
01945 gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01946 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01947 g_signal_handler_block (window->button_experiment,
01948                         window->id_experiment_name);
01949 gtk_combo_box_text_remove_all (window->combo_experiment);
01950 for (i = 0; i < input->nexperiments; ++i)
01951     gtk_combo_box_text_append_text (window->combo_experiment,
01952                                     input->experiment[i].name);
01953 g_signal_handler_unblock
01954 (window->button_experiment, window->
id_experiment_name);
01955 g_signal_handler_unblock (window->combo_experiment,
01956                         window->id_experiment);
01957 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01958 g_signal_handler_block (window->combo_variable, window->
id_variable);
01959 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01959 gtk_combo_box_text_remove_all (window->combo_variable);
01960 for (i = 0; i < input->nvariables; ++i)
01961     gtk_combo_box_text_append_text (window->combo_variable,
01962                                     input->variable[i].name);
01963 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01964 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01965 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01966 window_set_variable ();
01967 window_update ();
01968
01969 #if DEBUG_INTERFACE
01970 fprintf (stderr, "window_read: end\n");
01971 #endif
01972 return 1;
01973 }

```

Here is the call graph for this function:



4.13.2.22 window_remove_experiment()

```
void window_remove_experiment ( )
```

Function to remove an experiment in the main window.

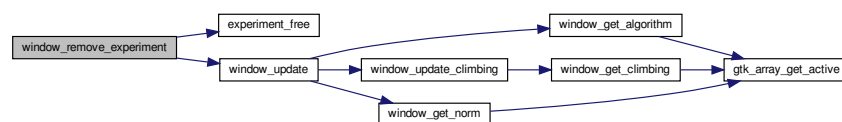
Definition at line 1355 of file [interface.c](#).

```

01356 {
01357     unsigned int i, j;
01358     #if DEBUG_INTERFACE
01359     fprintf (stderr, "window_remove_experiment: start\n");
01360     #endif
01361     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01362     g_signal_handler_block (window->combo_experiment, window->
        id_experiment);
01363     gtk_combo_box_text_remove (window->combo_experiment, i);
01364     g_signal_handler_unblock (window->combo_experiment,
        window->id_experiment);
01365     experiment_free (input->experiment + i, input->
        type);
01366     --input->nexperiments;
01367     for (j = i; j < input->nexperiments; ++j)
01368         memcpy (input->experiment + j, input->experiment + j + 1,
01369             sizeof (Experiment));
01370     j = input->nexperiments - 1;
01371     if (i > j)
01372         i = j;
01373     for (j = 0; j < input->experiment->ninputs; ++j)
01374         g_signal_handler_block (window->button_template[j],
        window->id_input[j]);
01375     g_signal_handler_block
01376         (window->button_experiment, window->
        id_experiment_name);
01377     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01378     g_signal_handler_unblock
01379         (window->button_experiment, window->
        id_experiment_name);
01380     for (j = 0; j < input->experiment->ninputs; ++j)
01381         g_signal_handler_unblock (window->button_template[j],
        window->id_input[j]);
01382     window_update ();
01383     #if DEBUG_INTERFACE
01384     fprintf (stderr, "window_remove_experiment: end\n");
01385     #endif
01386 }

```

Here is the call graph for this function:



4.13.2.23 window_remove_variable()

```
void window_remove_variable ( )
```

Function to remove a variable in the main window.

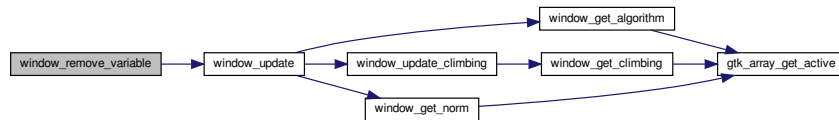
Definition at line 1625 of file [interface.c](#).

```

01626 {
01627     unsigned int i, j;
01628     #if DEBUG_INTERFACE
01629     fprintf (stderr, "window_remove_variable: start\n");
01630     #endif
01631     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01632     g_signal_handler_block (window->combo_variable, window->
id_variable);
01633     gtk_combo_box_text_remove (window->combo_variable, i);
01634     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01635     xmlFree (input->variable[i].name);
01636     --input->nvariables;
01637     for (j = i; j < input->nvariables; ++j)
01638         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01639     j = input->nvariables - 1;
01640     if (i > j)
01641         i = j;
01642     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01643     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01644     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01645     window_update ();
01646     #if DEBUG_INTERFACE
01647     fprintf (stderr, "window_remove_variable: end\n");
01648     #endif
01649 }

```

Here is the call graph for this function:



4.13.2.24 window_run()

```
void window_run ( )
```

Function to run a optimization.

Definition at line 974 of file [interface.c](#).

```

00975 {
00976     unsigned int i;
00977     char *msg, *msg2, buffer[64], buffer2[64];
00978     #if DEBUG_INTERFACE
00979     fprintf (stderr, "window_run: start\n");
00980     #endif
00981     if (!window_save ())
00982     {
00983         #if DEBUG_INTERFACE
00984         fprintf (stderr, "window_run: end\n");
00985         #endif
00986         return;
00987     }
00988     running_new ();
00989     while (gtk_events_pending ())
00990         gtk_main_iteration ();
00991     optimize_open ();
00992     #if DEBUG_INTERFACE
00993     fprintf (stderr, "window_run: closing running dialog\n");
00994     #endif

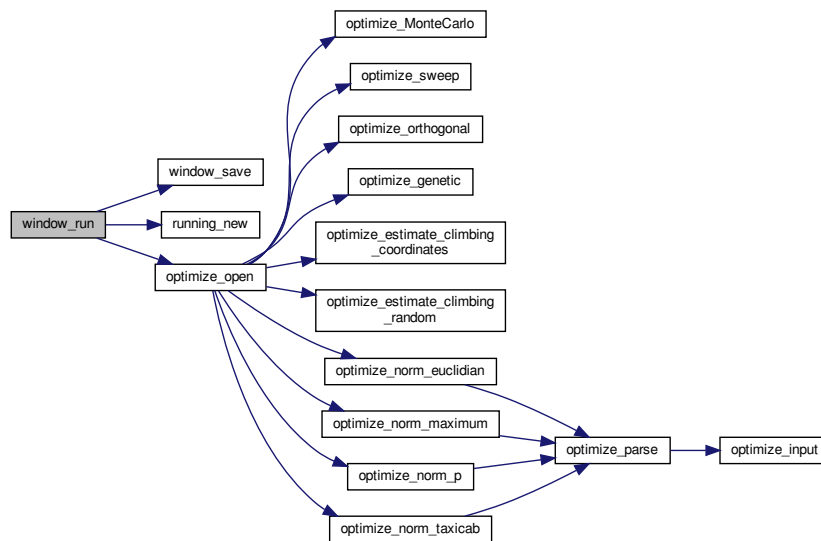
```

```

00995     gtk_spinner_stop (running->spinner);
00996     gtk_widget_destroy (GTK_WIDGET (running->dialog));
00997     #if DEBUG_INTERFACE
00998     fprintf (stderr, "window_run: displaying results\n");
00999     #endif
01000     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01001     msg2 = g_strdup (buffer);
01002     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01003     {
01004         snprintf (buffer, 64, "%s = %s\n",
01005             input->variable[i].name, format[input->
01006             variable[i].precision]);
01007         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008         msg = g_strconcat (msg2, buffer2, NULL);
01009         g_free (msg2);
01010     }
01010     snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01011         optimize->calculation_time);
01012     msg = g_strconcat (msg2, buffer, NULL);
01013     g_free (msg2);
01014     show_message (_("Best result"), msg, INFO_TYPE);
01015     g_free (msg);
01016     #if DEBUG_INTERFACE
01017     fprintf (stderr, "window_run: freeing memory\n");
01018     #endif
01019     optimize_free ();
01020     #if DEBUG_INTERFACE
01021     fprintf (stderr, "window_run: end\n");
01022     #endif
01023 }

```

Here is the call graph for this function:



4.13.2.25 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 823 of file [interface.c](#).

```

00824 {
00825     GtkFileChooserDialog *dlg;
00826     GtkFileFilter *filter1, *filter2;
00827     char *buffer;
00828
00829     #if DEBUG_INTERFACE
00830     fprintf (stderr, "window_save: start\n");
00831     #endif
00832
00833     // Opening the saving dialog
00834     dlg = (GtkFileChooserDialog *)
00835         gtk_file_chooser_dialog_new (_("Save file"),
00836                                     window->window,
00837                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00838                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00839                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00840     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00841     buffer = g_build_filename (input->directory, input->name, NULL);
00842     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00843     g_free (buffer);
00844
00845     // Adding XML filter
00846     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00847     gtk_file_filter_set_name (filter1, "XML");
00848     gtk_file_filter_add_pattern (filter1, "*.xml");
00849     gtk_file_filter_add_pattern (filter1, "*.XML");
00850     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00851
00852     // Adding JSON filter
00853     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00854     gtk_file_filter_set_name (filter2, "JSON");
00855     gtk_file_filter_add_pattern (filter2, "*.json");
00856     gtk_file_filter_add_pattern (filter2, "*.JSON");
00857     gtk_file_filter_add_pattern (filter2, "*.js");
00858     gtk_file_filter_add_pattern (filter2, "*.JS");
00859     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861     if (input->type == INPUT_TYPE_XML)
00862         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00863     else
00864         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00865
00866     // If OK response then saving
00867     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00868     {
00869         // Setting input file type
00870         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00871         buffer = (char *) gtk_file_filter_get_name (filter1);
00872         if (!strcmp (buffer, "XML"))
00873             input->type = INPUT_TYPE_XML;
00874         else
00875             input->type = INPUT_TYPE_JSON;
00876
00877         // Adding properties to the root XML node
00878         input->simulator = gtk_file_chooser_get_filename
00879             (GTK_FILE_CHOOSER (window->button_simulator));
00880         if (gtk_toggle_button_get_active
00881             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00882             input->evaluator = gtk_file_chooser_get_filename
00883                 (GTK_FILE_CHOOSER (window->button_evaluator));
00884         else
00885             input->evaluator = NULL;
00886         if (input->type == INPUT_TYPE_XML)
00887         {
00888             input->result
00889                 = (char *) xmlStrdup ((const xmlChar *)
00890                                     gtk_entry_get_text (window->entry_result));
00891             input->variables
00892                 = (char *) xmlStrdup ((const xmlChar *)
00893                                     gtk_entry_get_text (window->
00894                                     entry_variables));
00895         }
00896         else
00897         {
00898             input->result = g_strdup (gtk_entry_get_text (window->
00899             entry_result));
00898             input->variables =
00899                 g_strdup (gtk_entry_get_text (window->entry_variables));

```



```

00900     }
00901
00902     // Setting the algorithm
00903     switch (window_get_algorithm ())
00904     {
00905         case ALGORITHM_MONTE_CARLO:
00906             input->algorithm = ALGORITHM_MONTE_CARLO;
00907             input->nsimulations
00908                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00909             input->niterations
00910                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00911             input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00912             input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00913             window_save_climbing ();
00914             break;
00915         case ALGORITHM_SWEEP:
00916             input->algorithm = ALGORITHM_SWEEP;
00917             input->niterations
00918                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00919             input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00920             input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00921             window_save_climbing ();
00922             break;
00923         case ALGORITHM_ORTHOGONAL:
00924             input->algorithm = ALGORITHM_ORTHOGONAL;
00925             input->niterations
00926                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927             input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00928             input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00929             window_save_climbing ();
00930             break;
00931         default:
00932             input->algorithm = ALGORITHM_GENETIC;
00933             input->nsimulations
00934                 = gtk_spin_button_get_value_as_int (window->spin_population);
00935             input->niterations
00936                 = gtk_spin_button_get_value_as_int (window->spin_generations);
00937             input->mutation_ratio
00938                 = gtk_spin_button_get_value (window->spin_mutation);
00939             input->reproduction_ratio
00940                 = gtk_spin_button_get_value (window->spin_reproduction);
00941             input->adaptation_ratio
00942                 = gtk_spin_button_get_value (window->spin_adaptation);
00943             break;
00944     }
00945     input->norm = window_get_norm ();
00946     input->p = gtk_spin_button_get_value (window->spin_p);
00947     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00948
00949     // Saving the XML file
00950     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00951     input_save (buffer);
00952
00953     // Closing and freeing memory
00954     g_free (buffer);
00955     gtk_widget_destroy (GTK_WIDGET (dlg));
00956 #if DEBUG_INTERFACE
00957     fprintf (stderr, "window_save: end\n");
00958 #endif
00959     return 1;
00960 }
00961
00962 // Closing and freeing memory
00963 gtk_widget_destroy (GTK_WIDGET (dlg));
00964 #if DEBUG_INTERFACE
00965     fprintf (stderr, "window_save: end\n");
00966 #endif
00967     return 0;
00968 }

```

4.13.2.26 window_save_climbing()

```
void window_save_climbing ( )
```

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

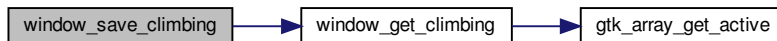
Definition at line 790 of file [interface.c](#).

```

00791 {
00792     #if DEBUG_INTERFACE
00793         fprintf (stderr, "window_save_climbing: start\n");
00794     #endif
00795     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
00796         check_climbing)))
00797     {
00798         input->nsteps = gtk_spin_button_get_value_as_int (window->
00799         spin_steps);
00800         input->relaxation = gtk_spin_button_get_value (window->
00801         spin_relaxation);
00802         switch (window_get_climbing ())
00803         {
00804             case CLIMBING_METHOD_COORDINATES:
00805                 input->climbing = CLIMBING_METHOD_COORDINATES;
00806                 break;
00807             default:
00808                 input->climbing = CLIMBING_METHOD_RANDOM;
00809                 input->nestimates
00810                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00811         }
00812     }
00813     else
00814         input->nsteps = 0;
00815     #if DEBUG_INTERFACE
00816         fprintf (stderr, "window_save_climbing: end\n");
00817     #endif
00818 }

```

Here is the call graph for this function:



4.13.2.27 window_set_algorithm()

```
void window_set_algorithm ( )
```

Function to avoid memory errors changing the algorithm.

Definition at line 1281 of file [interface.c](#).

```

01282 {
01283     int i;
01284     #if DEBUG_INTERFACE
01285         fprintf (stderr, "window_set_algorithm: start\n");
01286     #endif
01287     i = window_get_algorithm ();
01288     switch (i)
01289     {
01290         case ALGORITHM_SWEEP:
01291         case ALGORITHM_ORTHOGONAL:
01292             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01293             if (i < 0)
01294                 i = 0;
01295             gtk_spin_button_set_value (window->spin_sweeps,
01296                 (gdouble) input->variable[i].

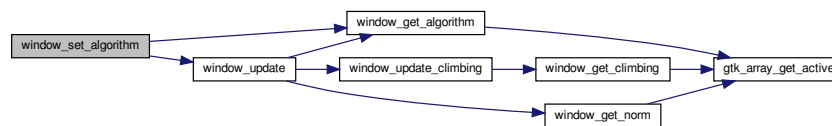
```

```

        nsweeps);
01297     break;
01298     case ALGORITHM_GENETIC:
01299         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01300         if (i < 0)
01301             i = 0;
01302         gtk_spin_button_set_value (window->spin_bits,
01303                                   (gdouble) input->variable[i].nbits);
01304     }
01305     window_update ();
01306 #if DEBUG_INTERFACE
01307     fprintf (stderr, "window_set_algorithm: end\n");
01308 #endif
01309 }

```

Here is the call graph for this function:



4.13.2.28 window_set_experiment()

```
void window_set_experiment ( )
```

Function to set the experiment data in the main window.

Definition at line 1315 of file [interface.c](#).

```

01316 {
01317     unsigned int i, j;
01318     char *buffer1, *buffer2;
01319 #if DEBUG_INTERFACE
01320     fprintf (stderr, "window_set_experiment: start\n");
01321 #endif
01322     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01323     gtk_spin_button_set_value (window->spin_weight, input->
01324                               experiment[i].weight);
01324     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01325     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01326     g_free (buffer1);
01327     g_signal_handler_block
01328         (window->button_experiment, window->
01329          id_experiment_name);
01329     gtk_file_chooser_set_filename
01330         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01331     g_signal_handler_unblock
01332         (window->button_experiment, window->
01333          id_experiment_name);
01333     g_free (buffer2);
01334     for (j = 0; j < input->experiment->ninputs; ++j)
01335     {
01336         g_signal_handler_block (window->button_template[j],
01337                                window->id_input[j]);
01337         buffer2 =
01338             g_build_filename (input->directory, input->experiment[i].
01339                               stencil[j],
01340                               NULL);
01340         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01341                                       (window->button_template[j]), buffer2);
01342         g_free (buffer2);
01343         g_signal_handler_unblock
01344             (window->button_template[j], window->id_input[j]);
01345     }
01346 #if DEBUG_INTERFACE
01347     fprintf (stderr, "window_set_experiment: end\n");
01348 #endif
01349 }

```

4.13.2.29 window_set_variable()

```
void window_set_variable ( )
```

Function to set the variable data in the main window.

Definition at line 1548 of file [interface.c](#).

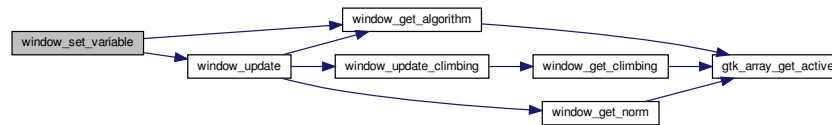
```
01549 {
01550     unsigned int i;
01551     #if DEBUG_INTERFACE
01552         fprintf (stderr, "window_set_variable: start\n");
01553     #endif
01554     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01555     g_signal_handler_block (window->entry_variable, window->
01556         id_variable_label);
01557     gtk_entry_set_text (window->entry_variable, input->
01558         variable[i].name);
01559     g_signal_handler_unblock (window->entry_variable, window->
01560         id_variable_label);
01561     gtk_spin_button_set_value (window->spin_min, input->
01562         variable[i].rangemin);
01563     gtk_spin_button_set_value (window->spin_max, input->
01564         variable[i].rangemax);
01565     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01566     {
01567         gtk_spin_button_set_value (window->spin_minabs,
01568             input->variable[i].rangeminabs);
01569         gtk_toggle_button_set_active
01570             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01571     }
01572     else
01573     {
01574         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01575         gtk_toggle_button_set_active
01576             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01577     }
01578     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01579     {
01580         gtk_spin_button_set_value (window->spin_maxabs,
01581             input->variable[i].rangemaxabs);
01582         gtk_toggle_button_set_active
01583             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01584     }
01585     else
01586     {
01587         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01588         gtk_toggle_button_set_active
01589             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01590     }
01591     gtk_spin_button_set_value (window->spin_precision,
01592         input->variable[i].precision);
01593     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01594         nsteps);
01595     if (input->nsteps)
01596         gtk_spin_button_set_value (window->spin_step, input->
01597             variable[i].step);
01598     #if DEBUG_INTERFACE
01599         fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01600             input->variable[i].precision);
01601     #endif
01602     switch (window_get_algorithm ())
01603     {
01604         case ALGORITHM_SWEEP:
01605         case ALGORITHM_ORTHOGONAL:
01606             gtk_spin_button_set_value (window->spin_sweeps,
01607                 (gdouble) input->variable[i].
01608                 nsweeps);
01609     #if DEBUG_INTERFACE
01610         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01611             input->variable[i].nsweeps);
01612     #endif
01613         break;
01614         case ALGORITHM_GENETIC:
01615             gtk_spin_button_set_value (window->spin_bits,
01616                 (gdouble) input->variable[i].nbits);
01617     #if DEBUG_INTERFACE
01618         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01619             input->variable[i].nbits);
01620     #endif
01621         break;
01622     }
```

```

01615     window_update ();
01616     #if DEBUG_INTERFACE
01617     fprintf (stderr, "window_set_variable: end\n");
01618     #endif
01619 }

```

Here is the call graph for this function:



4.13.2.30 window_template_experiment()

```

void window_template_experiment (
    void * data )

```

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

Parameters

<i>data</i>	Callback data (i-th input template).
-------------	--------------------------------------

Definition at line 1517 of file [interface.c](#).

```

01519 {
01520     unsigned int i, j;
01521     char *buffer;
01522     GFile *file1, *file2;
01523     #if DEBUG_INTERFACE
01524     fprintf (stderr, "window_template_experiment: start\n");
01525     #endif
01526     i = (size_t) data;
01527     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01528     file1
01529     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01530     file2 = g_file_new_for_path (input->directory);
01531     buffer = g_file_get_relative_path (file2, file1);
01532     if (input->type == INPUT_TYPE_XML)
01533         input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01534     else
01535         input->experiment[j].stencil[i] = g_strdup (buffer);
01536     g_free (buffer);
01537     g_object_unref (file2);
01538     g_object_unref (file1);
01539     #if DEBUG_INTERFACE
01540     fprintf (stderr, "window_template_experiment: end\n");
01541     #endif
01542 }

```

4.13.2.31 window_update()

```
void window_update ( )
```

Function to update the main window view.

Definition at line 1124 of file [interface.c](#).

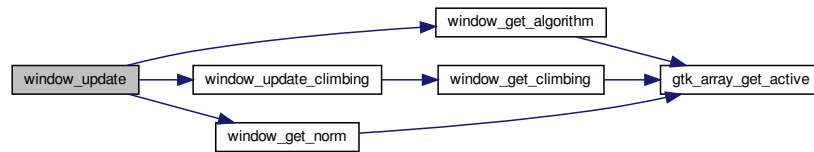
```
01125 {
01126     unsigned int i;
01127     #if DEBUG_INTERFACE
01128     fprintf (stderr, "window_update: start\n");
01129     #endif
01130     gtk_widget_set_sensitive
01131     (GTK_WIDGET (window->button_evaluator),
01132      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01133      (window->check_evaluator)));
01134     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01135     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01136     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01137     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01138     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01139     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01140     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01142     gtk_widget_hide (GTK_WIDGET (window->label_population));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01144     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01146     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01148     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01149     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01150     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01151     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01152     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01153     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01154     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01155     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01156     gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01157     gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01158     gtk_widget_hide (GTK_WIDGET (window->label_step));
01159     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01160     gtk_widget_hide (GTK_WIDGET (window->label_p));
01161     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01162     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01163     switch (window_get_algorithm ())
01164     {
01165     case ALGORITHM_MONTE_CARLO:
01166         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01167         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01168         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01169         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01170         if (i > 1)
01171         {
01172             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01173             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01174             gtk_widget_show (GTK_WIDGET (window->label_bests));
01175             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01176         }
01177         window_update_climbing ();
01178         break;
01179     case ALGORITHM_SWEEP:
01180     case ALGORITHM_ORTHOGONAL:
01181         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01182         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01183         if (i > 1)
01184         {
01185             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01186             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01187             gtk_widget_show (GTK_WIDGET (window->label_bests));
01188             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01189         }
01190         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01191         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01192         gtk_widget_show (GTK_WIDGET (window->check_climbing));
01193         window_update_climbing ();
01194         break;
01195     default:
01196         gtk_widget_show (GTK_WIDGET (window->label_population));
01197         gtk_widget_show (GTK_WIDGET (window->spin_population));
01198         gtk_widget_show (GTK_WIDGET (window->label_generations));
```

```

01199     gtk_widget_show (GTK_WIDGET (window->spin_generations));
01200     gtk_widget_show (GTK_WIDGET (window->label_mutation));
01201     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01202     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01203     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01204     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01205     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01206     gtk_widget_show (GTK_WIDGET (window->label_bits));
01207     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01208 }
01209 gtk_widget_set_sensitive
01210 (GTK_WIDGET (window->button_remove_experiment),
input->nexperiments > 1);
01211 gtk_widget_set_sensitive
01212 (GTK_WIDGET (window->button_remove_variable),
input->nvariables > 1);
01213 for (i = 0; i < input->experiment->ninputs; ++i)
01214 {
01215     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01216     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01217     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01218     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01219     g_signal_handler_block
01220     (window->check_template[i], window->
id_template[i]);
01221     g_signal_handler_block (window->button_template[i],
window->id_input[i]);
01222     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_template[i]), 1);
01223     g_signal_handler_unblock (window->button_template[i],
01224     window->id_input[i]);
01225     g_signal_handler_unblock (window->check_template[i],
01226     window->id_template[i]);
01227 }
01228 if (i > 0)
01229 {
01230     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01231     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01232     gtk_toggle_button_get_active
01233     GTK_TOGGLE_BUTTON (window->check_template
[i - 1]));
01234 }
01235 if (i < MAX_NINPUTS)
01236 {
01237     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01238     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01239     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01240     gtk_widget_set_sensitive
01241     (GTK_WIDGET (window->button_template[i]),
01242     gtk_toggle_button_get_active
01243     GTK_TOGGLE_BUTTON (window->check_template[i]));
01244     g_signal_handler_block
01245     (window->check_template[i], window->
id_template[i]);
01246     g_signal_handler_block (window->button_template[i],
window->id_input[i]);
01247     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_template[i]), 0);
01248     g_signal_handler_unblock (window->button_template[i],
01249     window->id_input[i]);
01250     g_signal_handler_unblock (window->check_template[i],
01251     window->id_template[i]);
01252 }
01253 while (++i < MAX_NINPUTS)
01254 {
01255     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01256     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01257 }
01258 gtk_widget_set_sensitive
01259 (GTK_WIDGET (window->spin_minabs),
01260     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01261 gtk_widget_set_sensitive
01262 (GTK_WIDGET (window->spin_maxabs),
01263     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01264 if (window_get_norm () == ERROR_NORM_P)
01265 {
01266     gtk_widget_show (GTK_WIDGET (window->label_p));
01267     gtk_widget_show (GTK_WIDGET (window->spin_p));
01268 }
01269 #if DEBUG_INTERFACE
01270 fprintf (stderr, "window_update: end\n");
01271 #endif
01272 }

```

Here is the call graph for this function:



4.13.2.32 window_update_climbing()

```
void window_update_climbing ( )
```

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

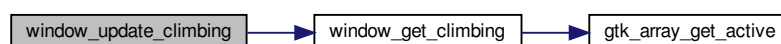
Definition at line 1093 of file [interface.c](#).

```

01094 {
01095     #if DEBUG_INTERFACE
01096         fprintf (stderr, "window_update_climbing: start\n");
01097     #endif
01098     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01099     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
01100         check_climbing)))
01101     {
01102         gtk_widget_show (GTK_WIDGET (window->grid_climbing));
01103         gtk_widget_show (GTK_WIDGET (window->label_step));
01104         gtk_widget_show (GTK_WIDGET (window->spin_step));
01105     }
01106     switch (window_get_climbing ())
01107     {
01108     case CLIMBING_METHOD_COORDINATES:
01109         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01110         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01111         break;
01112     default:
01113         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01114         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01115     }
01116     #if DEBUG_INTERFACE
01117     fprintf (stderr, "window_update_climbing: end\n");
01118     #endif
01119 }

```

Here is the call graph for this function:



4.13.2.33 window_update_variable()

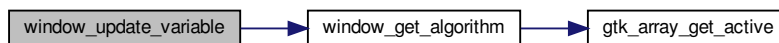
```
void window_update_variable ( )
```

Function to update the variable data in the main window.

Definition at line 1824 of file [interface.c](#).

```
01825 {
01826     int i;
01827     #if DEBUG_INTERFACE
01828     fprintf (stderr, "window_update_variable: start\n");
01829     #endif
01830     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01831     if (i < 0)
01832         i = 0;
01833     switch (window_get_algorithm ())
01834     {
01835     case ALGORITHM_SWEEP:
01836     case ALGORITHM_ORTHOGONAL:
01837         input->variable[i].nsweeps
01838         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01839     #if DEBUG_INTERFACE
01840         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01841                 input->variable[i].nsweeps);
01842     #endif
01843         break;
01844     case ALGORITHM_GENETIC:
01845         input->variable[i].nbits
01846         = gtk_spin_button_get_value_as_int (window->spin_bits);
01847     #if DEBUG_INTERFACE
01848         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01849                 input->variable[i].nbits);
01850     #endif
01851     }
01852     #if DEBUG_INTERFACE
01853     fprintf (stderr, "window_update_variable: end\n");
01854     #endif
01855 }
```

Here is the call graph for this function:



4.13.2.34 window_weight_experiment()

```
void window_weight_experiment ( )
```

Function to update the experiment weight in the main window.

Definition at line 1475 of file [interface.c](#).

```
01476 {
01477     unsigned int i;
01478     #if DEBUG_INTERFACE
01479     fprintf (stderr, "window_weight_experiment: start\n");
01480     #endif
01481     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01482     input->experiment[i].weight = gtk_spin_button_get_value (
01483         window->spin_weight);
01484     #if DEBUG_INTERFACE
01485     fprintf (stderr, "window_weight_experiment: end\n");
01486     #endif
01487 }
```

4.14 interface.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INTERFACE__H
00033 #define INTERFACE__H 1
00034
00035 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00036
00037 typedef struct
00038 {
00039     GtkWidget *dialog;
00040     GtkWidget *grid;
00041     GtkWidget *label_seed;
00042     GtkWidget *spin_seed;
00043     GtkWidget *label_threads;
00044     GtkWidget *spin_threads;
00045     GtkWidget *label_climbing;
00046     GtkWidget *spin_climbing;
00047 } Options;
00048
00049 typedef struct
00050 {
00051     GtkWidget *dialog;
00052     GtkWidget *label;
00053     GtkWidget *spinner;
00054     GtkWidget *grid;
00055 } Running;
00056
00057 typedef struct
00058 {
00059     GtkWidget *window;
00060     GtkWidget *grid;
00061     GtkWidget *bar_buttons;
00062     GtkWidget *button_open;
00063     GtkWidget *button_save;
00064     GtkWidget *button_run;
00065     GtkWidget *button_options;
00066     GtkWidget *button_help;
00067     GtkWidget *button_about;
00068     GtkWidget *button_exit;
00069     GtkWidget *grid_files;
00070     GtkWidget *label_simulator;
00071     GtkWidget *FileChooserButton *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *FileChooserButton *button_evaluator;
00074     GtkWidget *label_result;
00075     GtkWidget *entry_result;
00076     GtkWidget *label_variables;
00077     GtkWidget *entry_variables;
00078     GtkWidget *frame_norm;
00079     GtkWidget *grid_norm;
00080     GtkWidget *radio_button_norm[NNORMS];
00081     GtkWidget *label_p;
00082     GtkWidget *spin_p;
00083     GtkWidget *scrolled_p;
00084     GtkWidget *frame_algorithm;

```

```

00110   GtkWidget *grid_algorithm;
00111   GtkRadioButton *button_algorithm[NALGORITHMS];
00113   GtkLabel *label_simulations;
00114   GtkSpinButton *spin_simulations;
00116   GtkLabel *label_iterations;
00117   GtkSpinButton *spin_iterations;
00119   GtkLabel *label_tolerance;
00120   GtkSpinButton *spin_tolerance;
00121   GtkLabel *label_bests;
00122   GtkSpinButton *spin_bests;
00123   GtkLabel *label_population;
00124   GtkSpinButton *spin_population;
00126   GtkLabel *label_generations;
00127   GtkSpinButton *spin_generations;
00129   GtkLabel *label_mutation;
00130   GtkSpinButton *spin_mutation;
00131   GtkLabel *label_reproduction;
00132   GtkSpinButton *spin_reproduction;
00134   GtkLabel *label_adaptation;
00135   GtkSpinButton *spin_adaptation;
00137   GtkCheckButton *check_climbing;
00139   GtkWidget *grid_climbing;
00141   GtkRadioButton *button_climbing[NCLIMBINGS];
00143   GtkLabel *label_steps;
00144   GtkSpinButton *spin_steps;
00145   GtkLabel *label_estimates;
00146   GtkSpinButton *spin_estimates;
00148   GtkLabel *label_relaxation;
00150   GtkSpinButton *spin_relaxation;
00152   GtkLabel *label_threshold;
00153   GtkSpinButton *spin_threshold;
00154   GtkScrolledWindow *scrolled_threshold;
00156   GtkFrame *frame_variable;
00157   GtkWidget *grid_variable;
00158   GtkComboBoxText *combo_variable;
00160   GtkButton *button_add_variable;
00161   GtkButton *button_remove_variable;
00162   GtkLabel *label_variable;
00163   GtkEntry *entry_variable;
00164   GtkLabel *label_min;
00165   GtkSpinButton *spin_min;
00166   GtkScrolledWindow *scrolled_min;
00167   GtkLabel *label_max;
00168   GtkSpinButton *spin_max;
00169   GtkScrolledWindow *scrolled_max;
00170   GtkCheckButton *check_minabs;
00171   GtkSpinButton *spin_minabs;
00172   GtkScrolledWindow *scrolled_minabs;
00173   GtkCheckButton *check_maxabs;
00174   GtkSpinButton *spin_maxabs;
00175   GtkScrolledWindow *scrolled_maxabs;
00176   GtkLabel *label_precision;
00177   GtkSpinButton *spin_precision;
00178   GtkLabel *label_sweeps;
00179   GtkSpinButton *spin_sweeps;
00180   GtkLabel *label_bits;
00181   GtkSpinButton *spin_bits;
00182   GtkLabel *label_step;
00183   GtkSpinButton *spin_step;
00184   GtkScrolledWindow *scrolled_step;
00185   GtkFrame *frame_experiment;
00186   GtkWidget *grid_experiment;
00187   GtkComboBoxText *combo_experiment;
00188   GtkButton *button_add_experiment;
00189   GtkButton *button_remove_experiment;
00190   GtkLabel *label_experiment;
00191   GtkFileChooserButton *button_experiment;
00193   GtkLabel *label_weight;
00194   GtkSpinButton *spin_weight;
00195   GtkCheckButton *check_template[MAX_NINPUTS];
00197   GtkFileChooserButton *button_template[MAX_NINPUTS];
00199   GdkPixbuf *logo;
00200   Experiment *experiment;
00201   Variable *variable;
00202   char *application_directory;
00203   gulong id_experiment;
00204   gulong id_experiment_name;
00205   gulong id_variable;
00206   gulong id_variable_label;
00207   gulong id_template[MAX_NINPUTS];
00209   gulong id_input[MAX_NINPUTS];
00211   unsigned int nexperiments;
00212   unsigned int nvariables;
00213 } Window;
00214
00215 // Global variables
00216 extern const char *logo[];

```

```

00217 extern Options options[1];
00218 extern Running running[1];
00219 extern Window window[1];
00220
00221 // Inline functions
00222 #if GTK_MINOR_VERSION < 10
00223 static inline GtkWidget *
00224 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00225 {
00226     GtkWidget *button;
00227     GtkWidget *image;
00228     button = (GtkWidget *) gtk_button_new ();
00229     image = (GtkWidget *) gtk_image_new_from_icon_name (name, size);
00230     gtk_button_set_image (button, GTK_WIDGET (image));
00231     return button;
00232 }
00233 #endif
00234
00235 // Public functions
00236 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00237 void input_save (char *filename);
00238 void options_new ();
00239 void running_new ();
00240 unsigned int window_get_algorithm ();
00241 unsigned int window_get_climbing ();
00242 unsigned int window_get_norm ();
00243 void window_save_climbing ();
00244 int window_save ();
00245 void window_run ();
00246 void window_help ();
00247 void window_update_climbing ();
00248 void window_update ();
00249 void window_set_algorithm ();
00250 void window_set_experiment ();
00251 void window_remove_experiment ();
00252 void window_add_experiment ();
00253 void window_name_experiment ();
00254 void window_weight_experiment ();
00255 void window_inputs_experiment ();
00256 void window_template_experiment (void *data);
00257 void window_set_variable ();
00258 void window_remove_variable ();
00259 void window_add_variable ();
00260 void window_label_variable ();
00261 void window_precision_variable ();
00262 void window_rangemin_variable ();
00263 void window_rangemax_variable ();
00264 void window_rangeminabs_variable ();
00265 void window_rangemaxabs_variable ();
00266 void window_update_variable ();
00267 int window_read (char *filename);
00268 void window_open ();
00269 void window_new (GtkApplication * application);
00270
00271 #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 "utils.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 argc, char **argv)

4.15.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.16 main.c

```
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 2012-2018, 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 "utils.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
00071 int
00072 main (int argn, char **argc)
00073 {
00074     #if HAVE_GTK
00075         show_pending = process_pending;
00076     #endif
00077     return mpcotool (argn, argc);
00078 }

```

4.17 mpcotool.c File Reference

Main function 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 "utils.h"
#include "experiment.h"
#include "variable.h"

```


Parameters

<i>argn</i>	Arguments number.
<i>argc</i>	Arguments pointer.

Definition at line 79 of file [mpcotool.c](#).

```

00091 {
00092     #if HAVE_GTK
00093         GtkApplication *application;
00094         char *buffer;
00095     #endif
00096
00097     // Starting pseudo-random numbers generator
00098     #if DEBUG_MPCOTOOL
00099         fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00100     #endif
00101     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00102
00103     // Allowing spaces in the XML data file
00104     #if DEBUG_MPCOTOOL
00105         fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00106     #endif
00107     xmlKeepBlanksDefault (0);
00108
00109     // Starting MPI
00110     #if HAVE_MPI
00111     #if DEBUG_MPCOTOOL
00112         fprintf (stderr, "mpcotool: starting MPI\n");
00113     #endif
00114         MPI_Init (&argn, &argc);
00115         MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00116         MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00117         printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00118     #else
00119         ntasks = 1;
00120     #endif
00121
00122     // Resetting result and variables file names
00123     #if DEBUG_MPCOTOOL
00124         fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00125     #endif
00126     input->result = input->variables = NULL;
00127
00128     // Getting threads number and pseudo-random numbers generator seed
00129     nthreads_climbing = nthreads = cores_number ();
00130     optimize->seed = DEFAULT_RANDOM_SEED;
00131
00132     #if HAVE_GTK
00133
00134     // Setting local language and international floating point numbers notation
00135     setlocale (LC_ALL, "");
00136     setlocale (LC_NUMERIC, "C");
00137     window->application_directory = g_get_current_dir ();
00138     buffer = g_build_filename (window->application_directory,
00139                               LOCALE_DIR, NULL);
00139     bindtextdomain (PROGRAM_INTERFACE, buffer);
00140     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00141     textdomain (PROGRAM_INTERFACE);
00142
00143     // Initing GTK+
00144     gtk_disable_setlocale ();
00145     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00146                                       G_APPLICATION_FLAGS_NONE);
00147     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00148
00149     // Opening the main window
00150     g_application_run (G_APPLICATION (application), 0, NULL);
00151
00152     // Freeing memory
00153     input_free ();
00154     g_free (buffer);
00155     gtk_widget_destroy (GTK_WIDGET (window->window));
00156     g_object_unref (application);
00157     g_free (window->application_directory);
00158
00159     #else
00160
00161     // Checking syntax
00162     if (argn < 2)
00163         {

```



```

00164     printf ("The syntax is:\n"
00165             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00166             "[variables_file]\n");
00167     return 1;
00168 }
00169
00170 // Getting threads number and pseudo-random numbers generator seed
00171 #if DEBUG_MPCOTOOL
00172 fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
00173          "generator seed\n");
00174 #endif
00175 if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00176 {
00177     nthreads_climbing = nthreads = atoi (argc[2]);
00178     if (!nthreads)
00179     {
00180         printf ("Bad threads number\n");
00181         return 2;
00182     }
00183     argc += 2;
00184     argn -= 2;
00185     if (argn > 2 && !strcmp (argc[1], "-seed"))
00186     {
00187         optimize->seed = atoi (argc[2]);
00188         argc += 2;
00189         argn -= 2;
00190     }
00191 }
00192 else if (argn > 2 && !strcmp (argc[1], "-seed"))
00193 {
00194     optimize->seed = atoi (argc[2]);
00195     argc += 2;
00196     argn -= 2;
00197     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00198     {
00199         nthreads_climbing = nthreads = atoi (argc[2]);
00200         if (!nthreads)
00201         {
00202             printf ("Bad threads number\n");
00203             return 2;
00204         }
00205         argc += 2;
00206         argn -= 2;
00207     }
00208 }
00209 printf ("nthreads=%u\n", nthreads);
00210 printf ("seed=%lu\n", optimize->seed);
00211
00212 // Checking arguments
00213 #if DEBUG_MPCOTOOL
00214 fprintf (stderr, "mpcotool: checking arguments\n");
00215 #endif
00216 if (argn > 4 || argn < 2)
00217 {
00218     printf ("The syntax is:\n"
00219             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00220             "[variables_file]\n");
00221     return 1;
00222 }
00223 if (argn > 2)
00224     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00225 if (argn == 4)
00226     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00227
00228 // Making optimization
00229 #if DEBUG_MPCOTOOL
00230 fprintf (stderr, "mpcotool: making optimization\n");
00231 #endif
00232 if (input_open (argc[1]))
00233     optimize_open ();
00234
00235 // Freeing memory
00236 #if DEBUG_MPCOTOOL
00237 fprintf (stderr, "mpcotool: freeing memory and closing\n");
00238 #endif
00239 optimize_free ();
00240
00241 #endif
00242
00243 // Closing MPI
00244 #if HAVE_MPI
00245 MPI_Finalize ();
00246 #endif
00247
00248 // Freeing memory
00249 gsl_rng_free (optimize->rng);
00250

```

```

00251 // Closing
00252 return 0;
00253 }

```

4.18 mpcotool.c

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #if HAVE_GTK
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
00053 #endif
00054 #include "genetic/genetic.h"
00055 #include "utils.h"
00056 #include "experiment.h"
00057 #include "variable.h"
00058 #include "input.h"
00059 #include "optimize.h"
00060 #if HAVE_GTK
00061 #include "interface.h"
00062 #endif
00063 #include "mpcotool.h"
00064
00065 #define DEBUG_MPCOTOOL 0
00066
00067 int
00068 mpcotool (int argn
00069 #if HAVE_GTK
00070     __attribute__ ((unused))
00071 #endif
00072 ,
00073     char **argc
00074 #if HAVE_GTK
00075     __attribute__ ((unused))
00076 #endif

```

```

00088 #endif
00090 )
00091 {
00092 #if HAVE_GTK
00093     GtkApplication *application;
00094     char *buffer;
00095 #endif
00096
00097     // Starting pseudo-random numbers generator
00098 #if DEBUG_MPCOTOOL
00099     fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00100 #endif
00101     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00102
00103     // Allowing spaces in the XML data file
00104 #if DEBUG_MPCOTOOL
00105     fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00106 #endif
00107     xmlKeepBlanksDefault (0);
00108
00109     // Starting MPI
00110 #if HAVE_MPI
00111 #if DEBUG_MPCOTOOL
00112     fprintf (stderr, "mpcotool: starting MPI\n");
00113 #endif
00114     MPI_Init (&argn, &argc);
00115     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00116     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00117     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00118 #else
00119     ntasks = 1;
00120 #endif
00121
00122     // Resetting result and variables file names
00123 #if DEBUG_MPCOTOOL
00124     fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00125 #endif
00126     input->result = input->variables = NULL;
00127
00128     // Getting threads number and pseudo-random numbers generator seed
00129     nthreads_climbing = nthreads = cores_number ();
00130     optimize->seed = DEFAULT_RANDOM_SEED;
00131
00132 #if HAVE_GTK
00133
00134     // Setting local language and international floating point numbers notation
00135     setlocale (LC_ALL, "");
00136     setlocale (LC_NUMERIC, "C");
00137     window->application_directory = g_get_current_dir ();
00138     buffer = g_build_filename (window->application_directory,
00139                               LOCALE_DIR, NULL);
00139     bindtextdomain (PROGRAM_INTERFACE, buffer);
00140     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00141     textdomain (PROGRAM_INTERFACE);
00142
00143     // Initing GTK+
00144     gtk_disable_setlocale ();
00145     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00146                                       G_APPLICATION_FLAGS_NONE);
00147     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00148
00149     // Opening the main window
00150     g_application_run (G_APPLICATION (application), 0, NULL);
00151
00152     // Freeing memory
00153     input_free ();
00154     g_free (buffer);
00155     gtk_widget_destroy (GTK_WIDGET (window->window));
00156     g_object_unref (application);
00157     g_free (window->application_directory);
00158
00159 #else
00160
00161     // Checking syntax
00162     if (argn < 2)
00163     {
00164         printf ("The syntax is:\n"
00165                "    ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00166                "[variables_file]\n");
00167         return 1;
00168     }
00169
00170     // Getting threads number and pseudo-random numbers generator seed
00171 #if DEBUG_MPCOTOOL
00172     fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
00173            "generator seed\n");
00174 #endif

```

```

00175     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00176     {
00177         nthreads_climbing = nthreads = atoi (argc[2]);
00178         if (!nthreads)
00179         {
00180             printf ("Bad threads number\n");
00181             return 2;
00182         }
00183         argc += 2;
00184         argn -= 2;
00185         if (argn > 2 && !strcmp (argc[1], "-seed"))
00186         {
00187             optimize->seed = atoi (argc[2]);
00188             argc += 2;
00189             argn -= 2;
00190         }
00191     }
00192     else if (argn > 2 && !strcmp (argc[1], "-seed"))
00193     {
00194         optimize->seed = atoi (argc[2]);
00195         argc += 2;
00196         argn -= 2;
00197         if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00198         {
00199             nthreads_climbing = nthreads = atoi (argc[2]);
00200             if (!nthreads)
00201             {
00202                 printf ("Bad threads number\n");
00203                 return 2;
00204             }
00205             argc += 2;
00206             argn -= 2;
00207         }
00208     }
00209     printf ("nthreads=%u\n", nthreads);
00210     printf ("seed=%lu\n", optimize->seed);
00211
00212     // Checking arguments
00213     #if DEBUG_MPCOTOOL
00214     fprintf (stderr, "mpcotool: checking arguments\n");
00215     #endif
00216     if (argn > 4 || argn < 2)
00217     {
00218         printf ("The syntax is:\n"
00219             "    ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00220             "[variables_file]\n");
00221         return 1;
00222     }
00223     if (argn > 2)
00224         input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00225     if (argn == 4)
00226         input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00227
00228     // Making optimization
00229     #if DEBUG_MPCOTOOL
00230     fprintf (stderr, "mpcotool: making optimization\n");
00231     #endif
00232     if (input_open (argc[1]))
00233         optimize_open ();
00234
00235     // Freeing memory
00236     #if DEBUG_MPCOTOOL
00237     fprintf (stderr, "mpcotool: freeing memory and closing\n");
00238     #endif
00239     optimize_free ();
00240
00241     #endif
00242
00243     // Closing MPI
00244     #if HAVE_MPI
00245     MPI_Finalize ();
00246     #endif
00247
00248     // Freeing memory
00249     gsl_rng_free (optimize->rng);
00250
00251     // Closing
00252     return 0;
00253 }

```

4.19 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 "genetic/genetic.h"
#include "utils.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

- void [optimize_input](#) (unsigned int simulation, char *[input](#), GMappedFile *[stencil](#))
- double [optimize_parse](#) (unsigned int simulation, unsigned int experiment)
- double [optimize_norm_euclidian](#) (unsigned int simulation)
- double [optimize_norm_maximum](#) (unsigned int simulation)
- double [optimize_norm_p](#) (unsigned int simulation)
- double [optimize_norm_taxicab](#) (unsigned int simulation)
- void [optimize_print](#) ()
- void [optimize_save_variables](#) (unsigned int simulation, double error)
- void [optimize_best](#) (unsigned int simulation, double value)
- void [optimize_sequential](#) ()
- void * [optimize_thread](#) ([ParallelData](#) *data)
- void [optimize_merge](#) (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)
- void [optimize_synchronise](#) ()
- void [optimize_sweep](#) ()
- void [optimize_MonteCarlo](#) ()
- void [optimize_orthogonal](#) ()
- void [optimize_best_climbing](#) (unsigned int simulation, double value)
- void [optimize_climbing_sequential](#) (unsigned int simulation)

- void * [optimize_climbing_thread](#) (ParallelData *data)
- double [optimize_estimate_climbing_random](#) (unsigned int variable, unsigned int estimate)
- double [optimize_estimate_climbing_coordinates](#) (unsigned int variable, unsigned int estimate)
- void [optimize_step_climbing](#) (unsigned int simulation)
- void [optimize_climbing](#) ()
- double [optimize_genetic_objective](#) (**Entity** *entity)
- void [optimize_genetic](#) ()
- void [optimize_save_old](#) ()
- void [optimize_merge_old](#) ()
- void [optimize_refine](#) ()
- void [optimize_step](#) ()
- void [optimize_iterate](#) ()
- void [optimize_free](#) ()
- void [optimize_open](#) ()

Variables

- unsigned int [nthreads_climbing](#)
Number of threads for the hill climbing method.
- void(* [optimize_algorithm](#))()
Pointer to the function to perform a optimization algorithm step.
- double(* [optimize_estimate_climbing](#))(unsigned int variable, unsigned int estimate)
Pointer to the function to estimate the climbing.
- double(* [optimize_norm](#))(unsigned int simulation)
Pointer to the error norm function.
- [Optimize optimize](#) [1]
Optimization data.

4.19.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.19.2 Function Documentation

4.19.2.1 [optimize_best\(\)](#)

```
void optimize_best (
    unsigned int simulation,
    double value )
```

Function to save the best simulations.

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 444 of file [optimize.c](#).

```

00446 {
00447     unsigned int i, j;
00448     double e;
00449     #if DEBUG_OPTIMIZE
00450     fprintf (stderr, "optimize_best: start\n");
00451     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00452             optimize->nsaveds, optimize->nbest);
00453     #endif
00454     if (optimize->nsaveds < optimize->nbest
00455         || value < optimize->error_best[optimize->nsaveds - 1])
00456     {
00457         if (optimize->nsaveds < optimize->nbest)
00458             ++optimize->nsaveds;
00459         optimize->error_best[optimize->nsaveds - 1] = value;
00460         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00461         for (i = optimize->nsaveds; --i;)
00462         {
00463             if (optimize->error_best[i] < optimize->
00464                 error_best[i - 1])
00465             {
00466                 j = optimize->simulation_best[i];
00467                 e = optimize->error_best[i];
00468                 optimize->simulation_best[i] = optimize->
00469                     simulation_best[i - 1];
00470                 optimize->error_best[i] = optimize->
00471                     error_best[i - 1];
00472                 optimize->simulation_best[i - 1] = j;
00473                 optimize->error_best[i - 1] = e;
00474             }
00475             else
00476                 break;
00477         }
00478     }
00479     #if DEBUG_OPTIMIZE
00480     fprintf (stderr, "optimize_best: end\n");
00481     #endif
00482 }
```

4.19.2.2 optimize_best_climbing()

```

void optimize_best_climbing (
    unsigned int simulation,
    double value )
```

Function to save the best simulation in a hill climbing method.

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 806 of file [optimize.c](#).

```

00808 {
00809     #if DEBUG_OPTIMIZE
```

```

00810     fprintf (stderr, "optimize_best_climbing: start\n");
00811     fprintf (stderr,
00812             "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00813             simulation, value, optimize->error_best[0]);
00814 #endif
00815     if (value < optimize->error_best[0])
00816     {
00817         optimize->error_best[0] = value;
00818         optimize->simulation_best[0] = simulation;
00819 #if DEBUG_OPTIMIZE
00820         fprintf (stderr,
00821                 "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00822                 simulation, value);
00823 #endif
00824     }
00825 #if DEBUG_OPTIMIZE
00826     fprintf (stderr, "optimize_best_climbing: end\n");
00827 #endif
00828 }

```

4.19.2.3 optimize_climbing()

```
void optimize_climbing ( )
```

Function to optimize with a hill climbing method.

Definition at line 1034 of file [optimize.c](#).

```

01035 {
01036     unsigned int i, j, k, b, s, adjust;
01037 #if DEBUG_OPTIMIZE
01038     fprintf (stderr, "optimize_climbing: start\n");
01039 #endif
01040     for (i = 0; i < optimize->nvariables; ++i)
01041         optimize->climbing[i] = 0.;
01042     b = optimize->simulation_best[0] * optimize->
nvariables;
01043     s = optimize->nsimulations;
01044     adjust = 1;
01045     for (i = 0; i < optimize->nsteps; ++i, s += optimize->
nestimates, b = k)
01046     {
01047 #if DEBUG_OPTIMIZE
01048         fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01049                 i, optimize->simulation_best[0]);
01050 #endif
01051         optimize_step_climbing (s);
01052         k = optimize->simulation_best[0] * optimize->
nvariables;
01053 #if DEBUG_OPTIMIZE
01054         fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01055                 i, optimize->simulation_best[0]);
01056 #endif
01057         if (k == b)
01058         {
01059             if (adjust)
01060                 for (j = 0; j < optimize->nvariables; ++j)
01061                     optimize->step[j] *= 0.5;
01062             for (j = 0; j < optimize->nvariables; ++j)
01063                 optimize->climbing[j] = 0.;
01064             adjust = 1;
01065         }
01066         else
01067         {
01068             for (j = 0; j < optimize->nvariables; ++j)
01069             {
01070 #if DEBUG_OPTIMIZE
01071                 fprintf (stderr,
01072                         "optimize_climbing: best%u=%.14le old%u=%.14le\n",
01073                         j, optimize->value[k + j], j, optimize->
value[b + j]);
01074 #endif
01075                 optimize->climbing[j]
01076                     = (1. - optimize->relaxation) * optimize->
climbing[j]

```

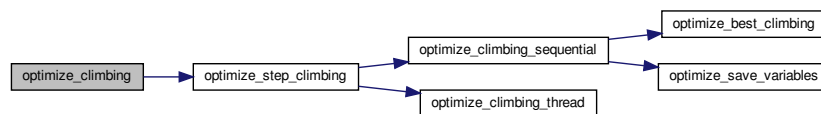


```

01077         + optimize->relaxation
01078         * (optimize->value[k + j] - optimize->value[b + j]);
01079 #if DEBUG_OPTIMIZE
01080     fprintf (stderr, "optimize_climbing: climbing%u=%.14le\n",
01081             j, optimize->climbing[j]);
01082 #endif
01083     }
01084     adjust = 0;
01085 }
01086 }
01087 #if DEBUG_OPTIMIZE
01088 fprintf (stderr, "optimize_climbing: end\n");
01089 #endif
01090 }

```

Here is the call graph for this function:



4.19.2.4 optimize_climbing_sequential()

```

void optimize_climbing_sequential (
    unsigned int simulation )

```

Function to estimate the hill climbing sequentially.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 834 of file [optimize.c](#).

```

00835 {
00836     double e;
00837     unsigned int i, j;
00838 #if DEBUG_OPTIMIZE
00839     fprintf (stderr, "optimize_climbing_sequential: start\n");
00840     fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00841             "nend_climbing=%u\n",
00842             optimize->nstart_climbing, optimize->
nend_climbing);
00843 #endif
00844     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00845     {
00846         j = simulation + i;
00847         e = optimize_norm (j);
00848         optimize_best_climbing (j, e);
00849         optimize_save_variables (j, e);
00850         if (e < optimize->threshold)
00851         {
00852             optimize->stop = 1;
00853             break;
00854         }
00855 #if DEBUG_OPTIMIZE
00856         fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
00857 #endif

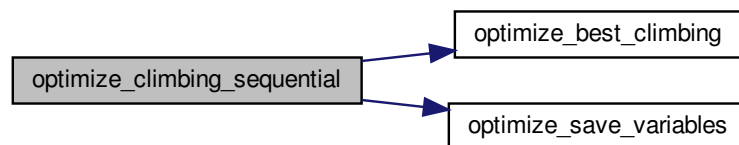
```

```

00858     }
00859     #if DEBUG_OPTIMIZE
00860     fprintf (stderr, "optimize_climbing_sequential: end\n");
00861     #endif
00862 }

```

Here is the call graph for this function:



4.19.2.5 optimize_climbing_thread()

```

void* optimize_climbing_thread (
    ParallelData * data )

```

Function to estimate the hill climbing on a thread.

Returns

NULL

Parameters

<i>data</i>	Function data.
-------------	----------------

Definition at line 870 of file [optimize.c](#).

```

00871 {
00872     unsigned int i, thread;
00873     double e;
00874     #if DEBUG_OPTIMIZE
00875     fprintf (stderr, "optimize_climbing_thread: start\n");
00876     #endif
00877     thread = data->thread;
00878     #if DEBUG_OPTIMIZE
00879     fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00880             thread,
00881             optimize->thread_climbing[thread],
00882             optimize->thread_climbing[thread + 1]);
00883     #endif
00884     for (i = optimize->thread_climbing[thread];
00885          i < optimize->thread_climbing[thread + 1]; ++i)
00886     {
00887         e = optimize_norm (i);
00888         g_mutex_lock (mutex);
00889         optimize_best_climbing (i, e);
00890         optimize_save_variables (i, e);

```

```

00891     if (e < optimize->threshold)
00892         optimize->stop = 1;
00893     g_mutex_unlock (mutex);
00894     if (optimize->stop)
00895         break;
00896 #if DEBUG_OPTIMIZE
00897     fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00898 #endif
00899 }
00900 #if DEBUG_OPTIMIZE
00901     fprintf (stderr, "optimize_climbing_thread: end\n");
00902 #endif
00903     g_thread_exit (NULL);
00904     return NULL;
00905 }

```

4.19.2.6 optimize_estimate_climbing_coordinates()

```

double optimize_estimate_climbing_coordinates (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the hill climbing vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 935 of file [optimize.c](#).

```

00939 {
00940     double x;
00941 #if DEBUG_OPTIMIZE
00942     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00943 #endif
00944     x = optimize->climbing[variable];
00945     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00946     {
00947         if (estimate & 1)
00948             x += optimize->step[variable];
00949         else
00950             x -= optimize->step[variable];
00951     }
00952 #if DEBUG_OPTIMIZE
00953     fprintf (stderr,
00954         "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
00955         variable, x);
00956     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00957 #endif
00958     return x;
00959 }

```

4.19.2.7 optimize_estimate_climbing_random()

```

double optimize_estimate_climbing_random (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the hill climbing vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 911 of file [optimize.c](#).

```

00916 {
00917     double x;
00918     #if DEBUG_OPTIMIZE
00919     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00920     #endif
00921     x = optimize->climbing[variable]
00922         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00923         step[variable];
00924     #if DEBUG_OPTIMIZE
00925     fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
00926             variable, x);
00927     #endif
00928     return x;
00929 }

```

4.19.2.8 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the [Optimize](#) struct.

Definition at line 1391 of file [optimize.c](#).

```

01392 {
01393     unsigned int i, j;
01394     #if DEBUG_OPTIMIZE
01395     fprintf (stderr, "optimize_free: start\n");
01396     #endif
01397     for (j = 0; j < optimize->ninputs; ++j)
01398     {
01399         for (i = 0; i < optimize->nexperiments; ++i)
01400             g_mapped_file_unref (optimize->file[j][i]);
01401         g_free (optimize->file[j]);
01402     }
01403     g_free (optimize->error_old);
01404     g_free (optimize->value_old);
01405     g_free (optimize->value);
01406     g_free (optimize->genetic_variable);
01407     #if DEBUG_OPTIMIZE
01408     fprintf (stderr, "optimize_free: end\n");
01409     #endif
01410 }

```

4.19.2.9 optimize_genetic()

```
void optimize_genetic ( )
```

Function to optimize with the genetic algorithm.

Definition at line 1131 of file [optimize.c](#).

```
01132 {
01133     double *best_variable = NULL;
01134     char *best_genome = NULL;
01135     double best_objective = 0.;
01136     #if DEBUG_OPTIMIZE
01137         fprintf (stderr, "optimize_genetic: start\n");
01138         fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01139                 nthreads);
01140         fprintf (stderr,
01141                 "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01142                 optimize->nvariables, optimize->
01143                 nsimulations, optimize->niterations);
01144         fprintf (stderr,
01145                 "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01146                 optimize->mutation_ratio, optimize->
01147                 reproduction_ratio,
01148                 optimize->adaptation_ratio);
01149     #endif
01150     genetic_algorithm_default (optimize->nvariables,
01151                               optimize->genetic_variable,
01152                               optimize->nsimulations,
01153                               optimize->niterations,
01154                               optimize->mutation_ratio,
01155                               optimize->reproduction_ratio,
01156                               optimize->adaptation_ratio,
01157                               optimize->seed,
01158                               optimize->threshold,
01159                               &optimize_genetic_objective,
01160                               &best_genome, &best_variable, &best_objective);
01161     #if DEBUG_OPTIMIZE
01162         fprintf (stderr, "optimize_genetic: the best\n");
01163     #endif
01164     optimize->error_old = (double *) g_malloc (sizeof (double));
01165     optimize->value_old
01166         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01167     optimize->error_old[0] = best_objective;
01168     memcpy (optimize->value_old, best_variable,
01169            optimize->nvariables * sizeof (double));
01170     g_free (best_genome);
01171     g_free (best_variable);
01172     optimize_print ();
01173     #if DEBUG_OPTIMIZE
01174         fprintf (stderr, "optimize_genetic: end\n");
01175     #endif
01176 }
```

4.19.2.10 optimize_genetic_objective()

```
double optimize_genetic_objective (
    Entity * entity )
```

Function to calculate the objective function of an entity.

Returns

objective function value.

Parameters

<i>entity</i>	entity data.
---------------	--------------

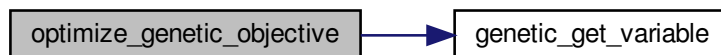
Definition at line 1098 of file [optimize.c](#).

```

01099 {
01100     unsigned int j;
01101     double objective;
01102     char buffer[64];
01103     #if DEBUG_OPTIMIZE
01104     fprintf (stderr, "optimize_genetic_objective: start\n");
01105     #endif
01106     for (j = 0; j < optimize->nvariables; ++j)
01107     {
01108         optimize->value[entity->id * optimize->nvariables + j]
01109         = genetic_get_variable (entity, optimize->genetic_variable + j);
01110     }
01111     objective = optimize_norm (entity->id);
01112     g_mutex_lock (mutex);
01113     for (j = 0; j < optimize->nvariables; ++j)
01114     {
01115         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01116         fprintf (optimize->file_variables, buffer,
01117                 genetic_get_variable (entity, optimize->genetic_variable + j));
01118     }
01119     fprintf (optimize->file_variables, "%.14le\n", objective);
01120     g_mutex_unlock (mutex);
01121     #if DEBUG_OPTIMIZE
01122     fprintf (stderr, "optimize_genetic_objective: end\n");
01123     #endif
01124     return objective;
01125 }

```

Here is the call graph for this function:



4.19.2.11 optimize_input()

```

void optimize_input (
    unsigned int simulation,
    char * input,
    GMappedFile * stencil )

```

Function to write the simulation input file.

Parameters

<i>simulation</i>	Simulation number.
<i>input</i>	Input file name.
<i>stencil</i>	Template of the input file name.

Definition at line 93 of file [optimize.c](#).

```

00096 {
00097     char buffer[32], value[32];
00098     GRegex *regex;
00099     FILE *file;
00100     char *buffer2, *buffer3 = NULL, *content;
00101     gsize length;
00102     unsigned int i;
00103
00104     #if DEBUG_OPTIMIZE
00105     fprintf (stderr, "optimize_input: start\n");
00106     #endif
00107
00108     // Checking the file
00109     if (!stencil)
00110         goto optimize_input_end;
00111
00112     // Opening stencil
00113     content = g_mapped_file_get_contents (stencil);
00114     length = g_mapped_file_get_length (stencil);
00115     #if DEBUG_OPTIMIZE
00116     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00117     #endif
00118     file = g_fopen (input, "w");
00119
00120     // Parsing stencil
00121     for (i = 0; i < optimize->nvariables; ++i)
00122     {
00123         #if DEBUG_OPTIMIZE
00124         fprintf (stderr, "optimize_input: variable=%u\n", i);
00125         #endif
00126         snprintf (buffer, 32, "@variable%u@", i + 1);
00127         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00128                             NULL);
00129         if (i == 0)
00130         {
00131             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00132                                               optimize->label[i],
00133                                               (GRegexMatchFlags) 0, NULL);
00134             #if DEBUG_OPTIMIZE
00135             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00136             #endif
00137         }
00138         else
00139         {
00140             length = strlen (buffer3);
00141             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00142                                               optimize->label[i],
00143                                               (GRegexMatchFlags) 0, NULL);
00144             g_free (buffer3);
00145         }
00146         g_regex_unref (regex);
00147         length = strlen (buffer2);
00148         snprintf (buffer, 32, "@value%u@", i + 1);
00149         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00150                             NULL);
00151         snprintf (value, 32, format[optimize->precision[i]],
00152                 optimize->value[simulation * optimize->
00153 nvariables + i]);
00154         #if DEBUG_OPTIMIZE
00155         fprintf (stderr, "optimize_input: value=%s\n", value);
00156         #endif
00157         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00158                                           (GRegexMatchFlags) 0, NULL);
00159         g_free (buffer2);
00160         g_regex_unref (regex);
00161     }
00162
00163     // Saving input file
00164     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00165     g_free (buffer3);
00166     fclose (file);
00167
00168 optimize_input_end:
00169     #if DEBUG_OPTIMIZE
00170     fprintf (stderr, "optimize_input: end\n");
00171     #endif
00172     return;
00173 }

```

4.19.2.12 optimize_iterate()

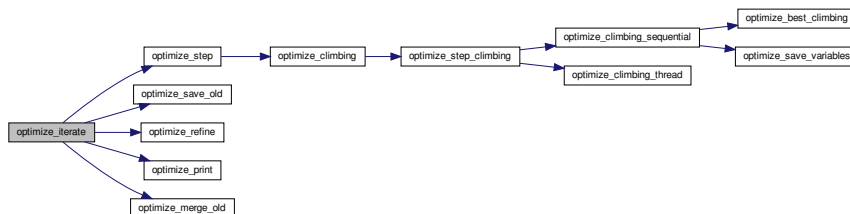
```
void optimize_iterate ( )
```

Function to iterate the algorithm.

Definition at line 1361 of file [optimize.c](#).

```
01362 {
01363     unsigned int i;
01364     #if DEBUG_OPTIMIZE
01365     fprintf (stderr, "optimize_iterate: start\n");
01366     #endif
01367     optimize->error_old = (double *) g_malloc (optimize->
nbest * sizeof (double));
01368     optimize->value_old =
01369     (double *) g_malloc (optimize->nbest * optimize->
nvariables *
01370     sizeof (double));
01371     optimize_step ();
01372     optimize_save_old ();
01373     optimize_refine ();
01374     optimize_print ();
01375     for (i = 1; i < optimize->niterations && !optimize->
stop; ++i)
01376     {
01377         optimize_step ();
01378         optimize_merge_old ();
01379         optimize_refine ();
01380         optimize_print ();
01381     }
01382     #if DEBUG_OPTIMIZE
01383     fprintf (stderr, "optimize_iterate: end\n");
01384     #endif
01385 }
```

Here is the call graph for this function:



4.19.2.13 optimize_merge()

```
void optimize_merge (
    unsigned int nsaveds,
    unsigned int * simulation_best,
    double * error_best )
```

Function to merge the 2 optimization results.

Parameters

<i>nsaveds</i>	Number of saved results.
<i>simulation_best</i>	Array of best simulation numbers.
<i>error_best</i>	Array of best objective function values.

Definition at line 557 of file [optimize.c](#).

```

00562 {
00563     unsigned int i, j, k, s[optimize->nbest];
00564     double e[optimize->nbest];
00565     #if DEBUG_OPTIMIZE
00566     fprintf (stderr, "optimize_merge: start\n");
00567     #endif
00568     i = j = k = 0;
00569     do
00570     {
00571         if (i == optimize->nsaveds)
00572         {
00573             s[k] = simulation_best[j];
00574             e[k] = error_best[j];
00575             ++j;
00576             ++k;
00577             if (j == nsaveds)
00578                 break;
00579         }
00580         else if (j == nsaveds)
00581         {
00582             s[k] = optimize->simulation_best[i];
00583             e[k] = optimize->error_best[i];
00584             ++i;
00585             ++k;
00586             if (i == optimize->nsaveds)
00587                 break;
00588         }
00589         else if (optimize->error_best[i] > error_best[j])
00590         {
00591             s[k] = simulation_best[j];
00592             e[k] = error_best[j];
00593             ++j;
00594             ++k;
00595         }
00596         else
00597         {
00598             s[k] = optimize->simulation_best[i];
00599             e[k] = optimize->error_best[i];
00600             ++i;
00601             ++k;
00602         }
00603     }
00604     while (k < optimize->nbest);
00605     optimize->nsaveds = k;
00606     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00607     memcpy (optimize->error_best, e, k * sizeof (double));
00608     #if DEBUG_OPTIMIZE
00609     fprintf (stderr, "optimize_merge: end\n");
00610     #endif
00611 }

```

4.19.2.14 optimize_merge_old()

```
void optimize_merge_old ( )
```

Function to merge the best results with the previous step best results on iterative methods.

Definition at line 1212 of file [optimize.c](#).

```

01213 {
01214     unsigned int i, j, k;
01215     double v[optimize->nbest * optimize->nvariables], e[
optimize->nbest],
01216         *enew, *eold;
01217     #if DEBUG_OPTIMIZE
01218     fprintf (stderr, "optimize_merge_old: start\n");
01219     #endif
01220     anew = optimize->error_best;
01221     eold = optimize->error_old;
01222     i = j = k = 0;
01223     do
01224     {
01225         if (*enew < *eold)
01226         {
01227             memcpy (v + k * optimize->nvariables,
optimize->value
01228                 + optimize->simulation_best[i] *
optimize->nvariables,
01229                     optimize->nvariables * sizeof (double));
01230             e[k] = *enew;
01231             ++k;
01232             ++enew;
01233             ++i;
01234         }
01235     }
01236     else
01237     {
01238         memcpy (v + k * optimize->nvariables,
optimize->value_old + j * optimize->
01239             nvariables,
optimize->nvariables * sizeof (double));
01240             e[k] = *eold;
01241             ++k;
01242             ++eold;
01243             ++j;
01244         }
01245     }
01246 }
01247 while (k < optimize->nbest);
01248 memcpy (optimize->value_old, v, k * optimize->
nvariables * sizeof (double));
01249 memcpy (optimize->error_old, e, k * sizeof (double));
01250 #if DEBUG_OPTIMIZE
01251 fprintf (stderr, "optimize_merge_old: end\n");
01252 #endif
01253 }

```

4.19.2.15 optimize_MonteCarlo()

```
void optimize_MonteCarlo ( )
```

Function to optimize with the Monte-Carlo algorithm.

Definition at line 715 of file [optimize.c](#).

```

00716 {
00717     unsigned int i, j;
00718     GThread *thread[nthreads];
00719     ParallelData data[nthreads];
00720     #if DEBUG_OPTIMIZE
00721     fprintf (stderr, "optimize_MonteCarlo: start\n");
00722     #endif
00723     for (i = 0; i < optimize->nsimulations; ++i)
00724     for (j = 0; j < optimize->nvariables; ++j)
00725         optimize->value[i * optimize->nvariables + j]
00726             = optimize->rangemin[j] + gsl_rng_uniform (optimize->
rng)
00727                 * (optimize->rangemax[j] - optimize->rangemin[j]);
00728     optimize->nsaveds = 0;
00729     if (nthreads <= 1)
00730         optimize_sequential ();
00731     else
00732     {
00733         for (i = 0; i < nthreads; ++i)
00734         {
00735             data[i].thread = i;

```

```

00736         thread[i]
00737         = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00738     }
00739     for (i = 0; i < nthreads; ++i)
00740         g_thread_join (thread[i]);
00741 }
00742 #if HAVE_MPI
00743 // Communicating tasks results
00744 optimize_synchronise ();
00745 #endif
00746 #if DEBUG_OPTIMIZE
00747 fprintf (stderr, "optimize_MonteCarlo: end\n");
00748 #endif
00749 }

```

4.19.2.16 optimize_norm_euclidian()

```
double optimize_norm_euclidian (
    unsigned int simulation )
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

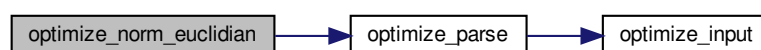
Definition at line 292 of file [optimize.c](#).

```

00293 {
00294     double e, ei;
00295     unsigned int i;
00296     #if DEBUG_OPTIMIZE
00297     fprintf (stderr, "optimize_norm_euclidian: start\n");
00298     #endif
00299     e = 0.;
00300     for (i = 0; i < optimize->nexperiments; ++i)
00301     {
00302         ei = optimize_parse (simulation, i);
00303         e += ei * ei;
00304     }
00305     e = sqrt (e);
00306     #if DEBUG_OPTIMIZE
00307     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00308     fprintf (stderr, "optimize_norm_euclidian: end\n");
00309     #endif
00310     return e;
00311 }

```

Here is the call graph for this function:



4.19.2.17 optimize_norm_maximum()

```
double optimize_norm_maximum (
    unsigned int simulation )
```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

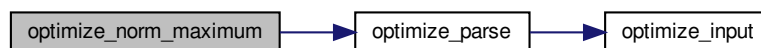
Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 319 of file [optimize.c](#).

```
00320 {
00321     double e, ei;
00322     unsigned int i;
00323     #if DEBUG_OPTIMIZE
00324     fprintf (stderr, "optimize_norm_maximum: start\n");
00325     #endif
00326     e = 0.;
00327     for (i = 0; i < optimize->nexperiments; ++i)
00328     {
00329         ei = fabs (optimize_parse (simulation, i));
00330         e = fmax (e, ei);
00331     }
00332     #if DEBUG_OPTIMIZE
00333     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00334     fprintf (stderr, "optimize_norm_maximum: end\n");
00335     #endif
00336     return e;
00337 }
```

Here is the call graph for this function:



4.19.2.18 optimize_norm_p()

```
double optimize_norm_p (
    unsigned int simulation )
```

Function to calculate the P error norm.

Returns

P error norm.

Parameters

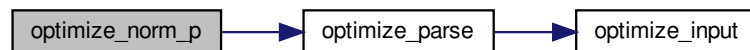
<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 345 of file [optimize.c](#).

```

00346 {
00347     double e, ei;
00348     unsigned int i;
00349     #if DEBUG_OPTIMIZE
00350     fprintf (stderr, "optimize_norm_p: start\n");
00351     #endif
00352     e = 0.;
00353     for (i = 0; i < optimize->nexperiments; ++i)
00354     {
00355         ei = fabs (optimize_parse (simulation, i));
00356         e += pow (ei, optimize->p);
00357     }
00358     e = pow (e, 1. / optimize->p);
00359     #if DEBUG_OPTIMIZE
00360     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00361     fprintf (stderr, "optimize_norm_p: end\n");
00362     #endif
00363     return e;
00364 }
```

Here is the call graph for this function:



4.19.2.19 optimize_norm_taxicab()

```

double optimize_norm_taxicab (
    unsigned int simulation )
```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 372 of file [optimize.c](#).

```

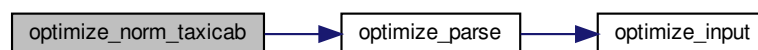
00373 {
```

```

00374     double e;
00375     unsigned int i;
00376     #if DEBUG_OPTIMIZE
00377     fprintf (stderr, "optimize_norm_taxicab: start\n");
00378     #endif
00379     e = 0.;
00380     for (i = 0; i < optimize->nexperiments; ++i)
00381         e += fabs (optimize_parse (simulation, i));
00382     #if DEBUG_OPTIMIZE
00383     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00384     fprintf (stderr, "optimize_norm_taxicab: end\n");
00385     #endif
00386     return e;
00387 }

```

Here is the call graph for this function:



4.19.2.20 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1416 of file [optimize.c](#).

```

01417 {
01418     GTimeZone *tz;
01419     GDateTime *t0, *t;
01420     unsigned int i, j;
01421
01422     #if DEBUG_OPTIMIZE
01423     char *buffer;
01424     fprintf (stderr, "optimize_open: start\n");
01425     #endif
01426
01427     // Getting initial time
01428     #if DEBUG_OPTIMIZE
01429     fprintf (stderr, "optimize_open: getting initial time\n");
01430     #endif
01431     tz = g_time_zone_new_utc ();
01432     t0 = g_date_time_new_now (tz);
01433
01434     // Obtaining and initing the pseudo-random numbers generator seed
01435     #if DEBUG_OPTIMIZE
01436     fprintf (stderr, "optimize_open: getting initial seed\n");
01437     #endif
01438     if (optimize->seed == DEFAULT_RANDOM_SEED)
01439         optimize->seed = input->seed;
01440     gsl_rng_set (optimize->rng, optimize->seed);
01441
01442     // Replacing the working directory
01443     #if DEBUG_OPTIMIZE
01444     fprintf (stderr, "optimize_open: replacing the working directory\n");
01445     #endif
01446     g_chdir (input->directory);
01447
01448     // Getting results file names
01449     optimize->result = input->result;
01450     optimize->variables = input->variables;

```

```

01451
01452 // Obtaining the simulator file
01453 optimize->simulator = input->simulator;
01454
01455 // Obtaining the evaluator file
01456 optimize->evaluator = input->evaluator;
01457
01458 // Reading the algorithm
01459 optimize->algorithm = input->algorithm;
01460 switch (optimize->algorithm)
01461 {
01462     case ALGORITHM_MONTE_CARLO:
01463         optimize_algorithm = optimize_MonteCarlo;
01464         break;
01465     case ALGORITHM_SWEEP:
01466         optimize_algorithm = optimize_sweep;
01467         break;
01468     case ALGORITHM_ORTHOGONAL:
01469         optimize_algorithm = optimize_orthogonal;
01470         break;
01471     default:
01472         optimize_algorithm = optimize_genetic;
01473         optimize->mutation_ratio = input->
mutation_ratio;
01474         optimize->reproduction_ratio = input->
reproduction_ratio;
01475         optimize->adaptation_ratio = input->
adaptation_ratio;
01476     }
01477     optimize->nvariables = input->nvariables;
01478     optimize->nsimulations = input->nsimulations;
01479     optimize->niterations = input->niterations;
01480     optimize->nbest = input->nbest;
01481     optimize->tolerance = input->tolerance;
01482     optimize->nsteps = input->nsteps;
01483     optimize->nestimates = 0;
01484     optimize->threshold = input->threshold;
01485     optimize->stop = 0;
01486     if (input->nsteps)
01487     {
01488         optimize->relaxation = input->relaxation;
01489         switch (input->climbing)
01490         {
01491             case CLIMBING_METHOD_COORDINATES:
01492                 optimize->nestimates = 2 * optimize->
nvariables;
01493                 optimize_estimate_climbing =
optimize_estimate_climbing_coordinates;
01494                 break;
01495             default:
01496                 optimize->nestimates = input->nestimates;
01497                 optimize_estimate_climbing =
optimize_estimate_climbing_random;
01498         }
01499     }
01500
01501 #if DEBUG_OPTIMIZE
01502     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01503 #endif
01504     optimize->simulation_best
01505     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01506     optimize->error_best = (double *) alloca (optimize->
nbest * sizeof (double));
01507
01508 // Reading the experimental data
01509 #if DEBUG_OPTIMIZE
01510     buffer = g_get_current_dir ();
01511     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01512     g_free (buffer);
01513 #endif
01514     optimize->nexperiments = input->nexperiments;
01515     optimize->ninputs = input->experiment->ninputs;
01516     optimize->experiment
01517     = (char **) alloca (input->nexperiments * sizeof (char *));
01518     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double
));
01519     for (i = 0; i < input->experiment->ninputs; ++i)
01520         optimize->file[i] = (GMappedFile **)
01521         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01522     for (i = 0; i < input->nexperiments; ++i)
01523     {
01524         #if DEBUG_OPTIMIZE
01525             fprintf (stderr, "optimize_open: i=%u\n", i);
01526         #endif
01527         optimize->experiment[i] = input->experiment[i].
name;
01528         optimize->weight[i] = input->experiment[i].

```

```

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

```



```

01606     fprintf (stderr, "optimize_open: allocating variables\n");
01607     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01608             optimize->nvariables, optimize->algorithm);
01609 #endif
01610     optimize->genetic_variable = NULL;
01611     if (optimize->algorithm == ALGORITHM_GENETIC)
01612     {
01613         optimize->genetic_variable = (GeneticVariable *)
01614             g_malloc (optimize->nvariables * sizeof (
01615             GeneticVariable));
01616         for (i = 0; i < optimize->nvariables; ++i)
01617         {
01618             #if DEBUG_OPTIMIZE
01619                 fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01620                     i, optimize->rangemin[i], optimize->
01621                     rangemax[i],
01622                     optimize->nbits[i]);
01623             #endif
01624             optimize->genetic_variable[i].minimum =
01625                 optimize->rangemin[i];
01626             optimize->genetic_variable[i].maximum =
01627                 optimize->rangemax[i];
01628             optimize->genetic_variable[i].nbits = optimize->
01629                 nbits[i];
01630         }
01631         #if DEBUG_OPTIMIZE
01632             fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01633                 optimize->nvariables, optimize->
01634                 nsimulations);
01635         #endif
01636         optimize->value = (double *)
01637             g_malloc ((optimize->nsimulations
01638                 + optimize->nestimates * optimize->
01639                 nsteps)
01640                 * optimize->nvariables * sizeof (double));
01641         // Calculating simulations to perform for each task
01642         #if HAVE_MPI
01643         #if DEBUG_OPTIMIZE
01644             fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01645                 optimize->mpi_rank, ntasks);
01646         #endif
01647         optimize->nstart = optimize->mpi_rank * optimize->
01648             nsimulations / ntasks;
01649         optimize->nend = (1 + optimize->mpi_rank) *
01650             optimize->nsimulations / ntasks;
01651         if (optimize->nsteps)
01652         {
01653             optimize->nstart_climbing
01654                 = optimize->mpi_rank * optimize->nestimates /
01655                 ntasks;
01656             optimize->nend_climbing
01657                 = (1 + optimize->mpi_rank) * optimize->
01658                 nestimates / ntasks;
01659         }
01660         #else
01661         optimize->nstart = 0;
01662         optimize->nend = optimize->nsimulations;
01663         if (optimize->nsteps)
01664         {
01665             optimize->nstart_climbing = 0;
01666             optimize->nend_climbing = optimize->
01667                 nestimates;
01668         }
01669         #endif
01670         #if DEBUG_OPTIMIZE
01671             fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->
01672                 nstart,
01673                 optimize->nend);
01674         #endif
01675         // Calculating simulations to perform for each thread
01676         optimize->thread
01677             = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01678         for (i = 0; i <= nthreads; ++i)
01679         {
01680             optimize->thread[i] = optimize->nstart
01681                 + i * (optimize->nend - optimize->nstart) / nthreads;
01682             #if DEBUG_OPTIMIZE
01683                 fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01684                     optimize->thread[i]);
01685             #endif
01686         }
01687         if (optimize->nsteps)
01688             optimize->thread_climbing = (unsigned int *)
01689                 alloca ((1 + nthreads_climbing) * sizeof (unsigned int));

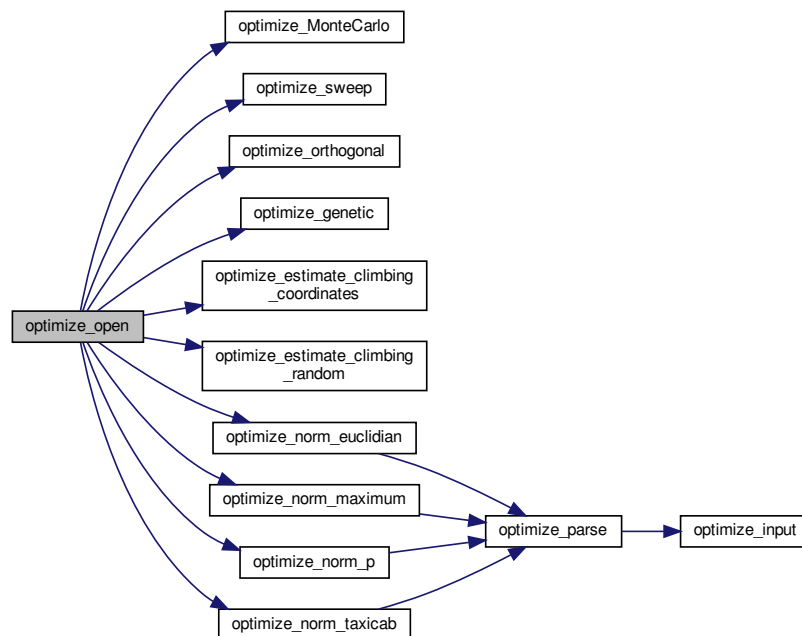
```

```

01680
01681 // Opening result files
01682 optimize->file_result = g_fopen (optimize->result, "w");
01683 optimize->file_variables = g_fopen (optimize->
variables, "w");
01684
01685 // Performing the algorithm
01686 switch (optimize->algorithm)
01687 {
01688     // Genetic algorithm
01689     case ALGORITHM_GENETIC:
01690         optimize_genetic ();
01691         break;
01692
01693     // Iterative algorithm
01694     default:
01695         optimize_iterate ();
01696 }
01697
01698 // Getting calculation time
01699 t = g_date_time_new_now (tz);
01700 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01701 g_date_time_unref (t);
01702 g_date_time_unref (t0);
01703 g_time_zone_unref (tz);
01704 printf ("%s = %.6lg s\n", _("Calculation time"), optimize->
calculation_time);
01705 fprintf (optimize->file_result, "%s = %.6lg s\n",
_ ("Calculation time"), optimize->calculation_time);
01706
01707
01708 // Closing result files
01709 fclose (optimize->file_variables);
01710 fclose (optimize->file_result);
01711
01712 #if DEBUG_OPTIMIZE
01713 fprintf (stderr, "optimize_open: end\n");
01714 #endif
01715 }

```

Here is the call graph for this function:



4.19.2.21 optimize_orthogonal()

```
void optimize_orthogonal ( )
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 755 of file [optimize.c](#).

```
00756 {
00757     unsigned int i, j, k, l;
00758     double e;
00759     GThread *thread[nthreads];
00760     ParallelData data[nthreads];
00761     #if DEBUG_OPTIMIZE
00762     fprintf (stderr, "optimize_orthogonal: start\n");
00763     #endif
00764     for (i = 0; i < optimize->nsimulations; ++i)
00765     {
00766         k = i;
00767         for (j = 0; j < optimize->nvariables; ++j)
00768         {
00769             l = k % optimize->nsweeps[j];
00770             k /= optimize->nsweeps[j];
00771             e = optimize->rangemin[j];
00772             if (optimize->nsweeps[j] > 1)
00773                 e += (1 + gsl_rng_uniform (optimize->rng))
00774                     * (optimize->rangemax[j] - optimize->
00775                        rangemin[j])
00776                     / optimize->nsweeps[j];
00777             optimize->value[i * optimize->nvariables + j] = e;
00778         }
00779         optimize->nsaveds = 0;
00780         if (nthreads <= 1)
00781             optimize_sequential ();
00782         else
00783         {
00784             for (i = 0; i < nthreads; ++i)
00785             {
00786                 data[i].thread = i;
00787                 thread[i]
00788                     = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00789             }
00790             for (i = 0; i < nthreads; ++i)
00791                 g_thread_join (thread[i]);
00792         }
00793         #if HAVE_MPI
00794         // Communicating tasks results
00795         optimize_synchronize ();
00796         #endif
00797         #if DEBUG_OPTIMIZE
00798         fprintf (stderr, "optimize_orthogonal: end\n");
00799         #endif
00800     }
```

4.19.2.22 optimize_parse()

```
double optimize_parse (
    unsigned int simulation,
    unsigned int experiment )
```

Function to parse input files, simulating and calculating the objective function.

Returns

Objective function value.

Parameters

<i>simulation</i>	Simulation number.
<i>experiment</i>	Experiment number.

Definition at line 182 of file [optimize.c](#).

```

00184 {
00185     unsigned int i;
00186     double e;
00187     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00188         *buffer3, *buffer4;
00189     FILE *file_result;
00190
00191     #if DEBUG_OPTIMIZE
00192     fprintf(stderr, "optimize_parse: start\n");
00193     fprintf(stderr, "optimize_parse: simulation=%u experiment=%u\n",
00194         simulation, experiment);
00195     #endif
00196
00197     // Opening input files
00198     for (i = 0; i < optimize->ninputs; ++i)
00199     {
00200         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00201         #if DEBUG_OPTIMIZE
00202         fprintf(stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00203         #endif
00204         optimize_input (simulation, &input[i][0], optimize->
00205             file[i][experiment]);
00206     }
00207     for (; i < MAX_NINPUTS; ++i)
00208         strcpy (&input[i][0], "");
00209     #if DEBUG_OPTIMIZE
00210     fprintf(stderr, "optimize_parse: parsing end\n");
00211     #endif
00212
00213     // Performing the simulation
00214     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00215     buffer2 = g_path_get_dirname (optimize->simulator);
00216     buffer3 = g_path_get_basename (optimize->simulator);
00217     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00218     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00219         buffer4, input[0], input[1], input[2], input[3], input[4],
00220         input[5], input[6], input[7], output);
00221     g_free (buffer4);
00222     g_free (buffer3);
00223     g_free (buffer2);
00224     #if DEBUG_OPTIMIZE
00225     fprintf(stderr, "optimize_parse: %s\n", buffer);
00226     #endif
00227     system (buffer);
00228
00229     // Checking the objective value function
00230     if (optimize->evaluator)
00231     {
00232         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00233         buffer2 = g_path_get_dirname (optimize->evaluator);
00234         buffer3 = g_path_get_basename (optimize->evaluator);
00235         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00236         snprintf (buffer, 512, "\"%s\" %s %s %s",
00237             buffer4, output, optimize->experiment[experiment], result);
00238         g_free (buffer4);
00239         g_free (buffer3);
00240         g_free (buffer2);
00241         #if DEBUG_OPTIMIZE
00242         fprintf(stderr, "optimize_parse: %s\n", buffer);
00243         #endif
00244         system (buffer);
00245         file_result = g_fopen (result, "r");
00246         e = atof (fgets (buffer, 512, file_result));
00247         fclose (file_result);
00248     }
00249     else
00250     {
00251         #if DEBUG_OPTIMIZE
00252         fprintf(stderr, "optimize_parse: output=%s\n", output);
00253         #endif
00254         strcpy (result, "");
00255         file_result = g_fopen (output, "r");
00256         e = atof (fgets (buffer, 512, file_result));

```

```

00257     fclose (file_result);
00258 }
00259
00260 // Removing files
00261 #if !DEBUG_OPTIMIZE
00262 for (i = 0; i < optimize->ninputs; ++i)
00263 {
00264     if (optimize->file[i][0])
00265     {
00266         snprintf (buffer, 512, RM " %s", &input[i][0]);
00267         system (buffer);
00268     }
00269 }
00270 snprintf (buffer, 512, RM " %s %s", output, result);
00271 system (buffer);
00272 #endif
00273
00274 // Processing pending events
00275 if (show_pending)
00276     show_pending ();
00277
00278 #if DEBUG_OPTIMIZE
00279 fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282 // Returning the objective function
00283 return e * optimize->weight[experiment];
00284 }

```

Here is the call graph for this function:



4.19.2.23 optimize_print()

```
void optimize_print ( )
```

Function to print the results.

Definition at line 393 of file [optimize.c](#).

```

00394 {
00395     unsigned int i;
00396     char buffer[512];
00397     #if HAVE_MPI
00398     if (optimize->mpi_rank)
00399         return;
00400     #endif
00401     printf ("%s\n", _("Best result"));
00402     fprintf (optimize->file_result, "%s\n", _("Best result"));
00403     printf ("error = %.15le\n", optimize->error_old[0]);
00404     fprintf (optimize->file_result, "error = %.15le\n",
00405             optimize->error_old[0]);
00406     for (i = 0; i < optimize->nvariables; ++i)
00407     {
00408         snprintf (buffer, 512, "%s = %s\n",
00409                 optimize->label[i], format[optimize->
00410 precision[i]]);
00409         printf (buffer, optimize->value_old[i]);
00410         fprintf (optimize->file_result, buffer, optimize->
00411 value_old[i]);
00411     }
00412     fflush (optimize->file_result);
00413 }

```

4.19.2.24 optimize_refine()

```
void optimize_refine ( )
```

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

Definition at line 1260 of file [optimize.c](#).

```
01261 {
01262     unsigned int i, j;
01263     double d;
01264     #if HAVE_MPI
01265     MPI_Status mpi_stat;
01266     #endif
01267     #if DEBUG_OPTIMIZE
01268     fprintf (stderr, "optimize_refine: start\n");
01269     #endif
01270     #if HAVE_MPI
01271     if (!optimize->mpi_rank)
01272     {
01273     #endif
01274         for (j = 0; j < optimize->nvariables; ++j)
01275         {
01276             optimize->rangemin[j] = optimize->rangemax[j]
01277             = optimize->value_old[j];
01278         }
01279         for (i = 0; ++i < optimize->nbest;)
01280         {
01281             for (j = 0; j < optimize->nvariables; ++j)
01282             {
01283                 optimize->rangemin[j]
01284                 = fmin (optimize->rangemin[j],
01285                     optimize->value_old[i * optimize->
01286                     nvariables + j]);
01287                 optimize->rangemax[j]
01288                 = fmax (optimize->rangemax[j],
01289                     optimize->value_old[i * optimize->
01290                     nvariables + j]);
01291             }
01292             for (j = 0; j < optimize->nvariables; ++j)
01293             {
01294                 d = optimize->tolerance
01295                 * (optimize->rangemax[j] - optimize->
01296                 rangemin[j]);
01297                 switch (optimize->algorithm)
01298                 {
01299                     case ALGORITHM_MONTE_CARLO:
01300                         d *= 0.5;
01301                         break;
01302                     default:
01303                         if (optimize->nsweeps[j] > 1)
01304                             d /= optimize->nsweeps[j] - 1;
01305                         else
01306                             d = 0.;
01307                 }
01308                 optimize->rangemin[j] -= d;
01309                 optimize->rangemin[j]
01310                 = fmax (optimize->rangemin[j], optimize->
01311                 rangeminabs[j]);
01312                 optimize->rangemax[j] += d;
01313                 optimize->rangemax[j]
01314                 = fmin (optimize->rangemax[j], optimize->
01315                 rangemaxabs[j]);
01316                 printf ("%s min=%lg max=%lg\n", optimize->label[j],
01317                     optimize->rangemin[j], optimize->
01318                     rangemax[j]);
01319                 fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01320                     optimize->label[j], optimize->rangemin[j],
01321                     optimize->rangemax[j]);
01322             }
01323             #if HAVE_MPI
01324             for (i = 1; i < ntasks; ++i)
01325             {
01326                 MPI_Send (optimize->rangemin, optimize->
01327                 nvariables, MPI_DOUBLE, i,
01328                 1, MPI_COMM_WORLD);
01329                 MPI_Send (optimize->rangemax, optimize->
01330                 nvariables, MPI_DOUBLE, i,
01331                 1, MPI_COMM_WORLD);
01332             }
01333         }
01334     }
01335     #endif
01336 }
```

```

01327     else
01328     {
01329         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
01330             1,
01331             MPI_COMM_WORLD, &mpi_stat);
01332         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
01333             1,
01334             MPI_COMM_WORLD, &mpi_stat);
01335     }
01336 #endif
01337 #if DEBUG_OPTIMIZE
01338     fprintf (stderr, "optimize_refine: end\n");
01339 #endif
01340 }

```

4.19.2.25 optimize_save_old()

```
void optimize_save_old ( )
```

Function to save the best results on iterative methods.

Definition at line 1180 of file [optimize.c](#).

```

01181 {
01182     unsigned int i, j;
01183     #if DEBUG_OPTIMIZE
01184         fprintf (stderr, "optimize_save_old: start\n");
01185         fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01186     #endif
01187     memcpy (optimize->error_old, optimize->error_best,
01188         optimize->nbest * sizeof (double));
01189     for (i = 0; i < optimize->nbest; ++i)
01190     {
01191         j = optimize->simulation_best[i];
01192     #if DEBUG_OPTIMIZE
01193         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01194     #endif
01195     }
01196     memcpy (optimize->value_old + i * optimize->
01197         nvariables,
01198         optimize->value + j * optimize->nvariables,
01199         optimize->nvariables * sizeof (double));
01200     }
01201 #if DEBUG_OPTIMIZE
01202     for (i = 0; i < optimize->nvariables; ++i)
01203         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01204             i, optimize->value_old[i]);
01205     fprintf (stderr, "optimize_save_old: end\n");
01206 #endif
01207 }

```

4.19.2.26 optimize_save_variables()

```
void optimize_save_variables (
    unsigned int simulation,
    double error )
```

Function to save in a file the variables and the error.

Parameters

<i>simulation</i>	Simulation number.
<i>error</i>	Error value.

Definition at line 419 of file `optimize.c`.

```

00421 {
00422     unsigned int i;
00423     char buffer[64];
00424     #if DEBUG_OPTIMIZE
00425     fprintf (stderr, "optimize_save_variables: start\n");
00426     #endif
00427     for (i = 0; i < optimize->nvariables; ++i)
00428     {
00429         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00430         fprintf (optimize->file_variables, buffer,
00431                 optimize->value[simulation * optimize->
nvariables + i]);
00432     }
00433     fprintf (optimize->file_variables, "%.14le\n", error);
00434     fflush (optimize->file_variables);
00435     #if DEBUG_OPTIMIZE
00436     fprintf (stderr, "optimize_save_variables: end\n");
00437     #endif
00438 }

```

4.19.2.27 optimize_sequential()

```
void optimize_sequential ( )
```

Function to optimize sequentially.

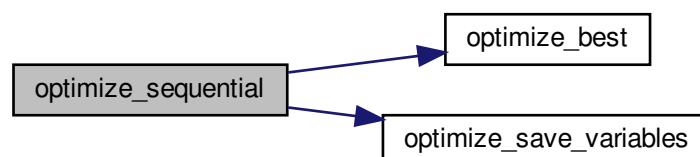
Definition at line 485 of file `optimize.c`.

```

00486 {
00487     unsigned int i;
00488     double e;
00489     #if DEBUG_OPTIMIZE
00490     fprintf (stderr, "optimize_sequential: start\n");
00491     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00492             optimize->nstart, optimize->nend);
00493     #endif
00494     for (i = optimize->nstart; i < optimize->nend; ++i)
00495     {
00496         e = optimize_norm (i);
00497         optimize_best (i, e);
00498         optimize_save_variables (i, e);
00499         if (e < optimize->threshold)
00500         {
00501             optimize->stop = 1;
00502             break;
00503         }
00504     #if DEBUG_OPTIMIZE
00505     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00506     #endif
00507     }
00508     #if DEBUG_OPTIMIZE
00509     fprintf (stderr, "optimize_sequential: end\n");
00510     #endif
00511 }

```

Here is the call graph for this function:



4.19.2.28 optimize_step()

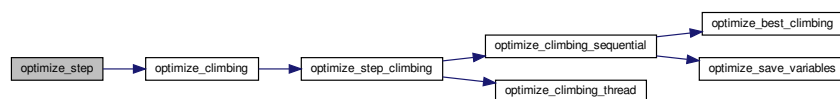
```
void optimize_step ( )
```

Function to do a step of the iterative algorithm.

Definition at line 1344 of file [optimize.c](#).

```
01345 {
01346     #if DEBUG_OPTIMIZE
01347         fprintf (stderr, "optimize_step: start\n");
01348     #endif
01349     optimize_algorithm ();
01350     if (optimize->nsteps)
01351         optimize_climbing ();
01352     #if DEBUG_OPTIMIZE
01353         fprintf (stderr, "optimize_step: end\n");
01354     #endif
01355 }
```

Here is the call graph for this function:



4.19.2.29 optimize_step_climbing()

```
void optimize_step_climbing (
    unsigned int simulation )
```

Function to do a step of the hill climbing method.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 965 of file [optimize.c](#).

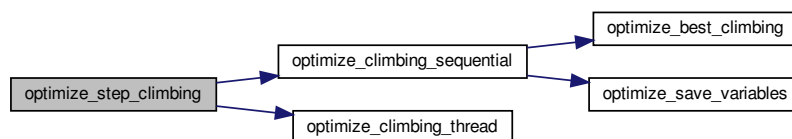
```
00966 {
00967     GThread *thread[nthreads_climbing];
00968     ParallelData data[nthreads_climbing];
00969     unsigned int i, j, k, b;
00970     #if DEBUG_OPTIMIZE
00971         fprintf (stderr, "optimize_step_climbing: start\n");
00972     #endif
00973     for (i = 0; i < optimize->nestimates; ++i)
00974     {
```

```

00975     k = (simulation + i) * optimize->nvariables;
00976     b = optimize->simulation_best[0] * optimize->
nvariables;
00977 #if DEBUG_OPTIMIZE
00978     fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
00979             simulation + i, optimize->simulation_best[0]);
00980 #endif
00981     for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00982     {
00983 #if DEBUG_OPTIMIZE
00984         fprintf (stderr,
00985                 "optimize_step_climbing: estimate=%u best%u=%.14le\n",
00986                 i, j, optimize->value[b]);
00987 #endif
00988         optimize->value[k]
00989             = optimize->value[b] + optimize_estimate_climbing (j, i)
;
00990         optimize->value[k] = fmin (fmax (optimize->value[k],
00991                                         optimize->rangeminabs[j]),
00992                                   optimize->rangemaxabs[j]);
00993 #if DEBUG_OPTIMIZE
00994         fprintf (stderr,
00995                 "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
00996                 i, j, optimize->value[k]);
00997 #endif
00998     }
00999 }
01000 if (nthreads_climbing == 1)
01001     optimize_climbing_sequential (simulation);
01002 else
01003 {
01004     for (i = 0; i <= nthreads_climbing; ++i)
01005     {
01006         optimize->thread_climbing[i]
01007             = simulation + optimize->nstart_climbing
01008             + i * (optimize->nend_climbing - optimize->
nstart_climbing)
01009             / nthreads_climbing;
01010 #if DEBUG_OPTIMIZE
01011         fprintf (stderr,
01012                 "optimize_step_climbing: i=%u thread_climbing=%u\n",
01013                 i, optimize->thread_climbing[i]);
01014 #endif
01015     }
01016     for (i = 0; i < nthreads_climbing; ++i)
01017     {
01018         data[i].thread = i;
01019         thread[i] = g_thread_new
01020             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01021     }
01022     for (i = 0; i < nthreads_climbing; ++i)
01023         g_thread_join (thread[i]);
01024 }
01025 #if DEBUG_OPTIMIZE
01026 fprintf (stderr, "optimize_step_climbing: end\n");
01027 #endif
01028 }

```

Here is the call graph for this function:



4.19.2.30 optimize_sweep()

```
void optimize_sweep ( )
```

Function to optimize with the sweep algorithm.

Definition at line 665 of file [optimize.c](#).

```

00666 {
00667     unsigned int i, j, k, l;
00668     double e;
00669     GThread *thread[nthreads];
00670     ParallelData data[nthreads];
00671     #if DEBUG_OPTIMIZE
00672     fprintf (stderr, "optimize_sweep: start\n");
00673     #endif
00674     for (i = 0; i < optimize->nsimulations; ++i)
00675     {
00676         k = i;
00677         for (j = 0; j < optimize->nvariables; ++j)
00678         {
00679             l = k % optimize->nsweeps[j];
00680             k /= optimize->nsweeps[j];
00681             e = optimize->rangemin[j];
00682             if (optimize->nsweeps[j] > 1)
00683                 e += 1 * (optimize->rangemax[j] - optimize->
rangemin[j])
00684                     / (optimize->nsweeps[j] - 1);
00685             optimize->value[i * optimize->nvariables + j] = e;
00686         }
00687     }
00688     optimize->nsaveds = 0;
00689     if (nthreads <= 1)
00690         optimize_sequential ();
00691     else
00692     {
00693         for (i = 0; i < nthreads; ++i)
00694         {
00695             data[i].thread = i;
00696             thread[i]
= g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00698         }
00699         for (i = 0; i < nthreads; ++i)
00700             g_thread_join (thread[i]);
00701     }
00702     #if HAVE_MPI
00703     // Communicating tasks results
00704     optimize_synchronise ();
00705     #endif
00706     #if DEBUG_OPTIMIZE
00707     fprintf (stderr, "optimize_sweep: end\n");
00708     #endif
00709 }

```

4.19.2.31 optimize_synchronise()

```
void optimize_synchronise ( )
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 618 of file [optimize.c](#).

```

00619 {
00620     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00621     double error_best[optimize->nbest];
00622     MPI_Status mpi_stat;
00623     #if DEBUG_OPTIMIZE
00624     fprintf (stderr, "optimize_synchronise: start\n");
00625     #endif
00626     if (optimize->mpi_rank == 0)
00627     {
00628         for (i = 1; i < ntasks; ++i)
00629         {
00630             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00631             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
MPI_COMM_WORLD, &mpi_stat);
00632

```

```

00633         MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00634                   MPI_COMM_WORLD, &mpi_stat);
00635         optimize_merge (nsaveds, simulation_best, error_best);
00636         MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637         if (stop)
00638             optimize->stop = 1;
00639     }
00640     for (i = 1; i < ntasks; ++i)
00641         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00642 }
00643 else
00644 {
00645     MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00646     MPI_Send (optimize->simulation_best, optimize->
00647 nsaveds, MPI_INT, 0, 1,
00648 MPI_COMM_WORLD);
00649     MPI_Send (optimize->error_best, optimize->
00650 nsaveds, MPI_DOUBLE, 0, 1,
00651 MPI_COMM_WORLD);
00652     MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00653     MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00654     if (stop)
00655         optimize->stop = 1;
00656 }
00657 #if DEBUG_OPTIMIZE
00658 fprintf (stderr, "optimize_synchronise: end\n");
00659 #endif
00660 }

```

4.19.2.32 optimize_thread()

```

void* optimize_thread (
    ParallelData * data )

```

Function to optimize on a thread.

Returns

NULL.

Parameters

<i>data</i>	Function data.
-------------	----------------

Definition at line 519 of file [optimize.c](#).

```

00520 {
00521     unsigned int i, thread;
00522     double e;
00523     #if DEBUG_OPTIMIZE
00524     fprintf (stderr, "optimize_thread: start\n");
00525     #endif
00526     thread = data->thread;
00527     #if DEBUG_OPTIMIZE
00528     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00529 optimize->thread[thread], optimize->thread[thread + 1]);
00530     #endif
00531     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00532     {
00533         e = optimize_norm (i);
00534         g_mutex_lock (mutex);
00535         optimize_best (i, e);
00536         optimize_save_variables (i, e);
00537         if (e < optimize->threshold)
00538             optimize->stop = 1;
00539         g_mutex_unlock (mutex);

```

```

00540         if (optimize->stop)
00541             break;
00542 #if DEBUG_OPTIMIZE
00543     fprintf(stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00544 #endif
00545     }
00546 #if DEBUG_OPTIMIZE
00547     fprintf(stderr, "optimize_thread: end\n");
00548 #endif
00549     g_thread_exit (NULL);
00550     return NULL;
00551 }

```

4.20 optimize.c

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <sys/param.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <glib/gstdio.h>
00044 #include <json-glib/json-glib.h>
00045 #ifdef G_OS_WIN32
00046 #include <windows.h>
00047 #elif !defined(__BSD_VISIBLE) && !defined(NETBSD)
00048 #include <alloca.h>
00049 #endif
00050 #if HAVE_MPI
00051 #include <mpi.h>
00052 #endif
00053 #include "genetic/genetic.h"
00054 #include "utils.h"
00055 #include "experiment.h"
00056 #include "variable.h"
00057 #include "input.h"
00058 #include "optimize.h"
00059
00060 #define DEBUG_OPTIMIZE 0
00061
00062 #ifdef G_OS_WIN32
00063 #define RM "del"
00064 #else
00065 #define RM "rm"
00066 #endif

```

```

00077
00078 unsigned int nthreads_climbing;
00080 void (*optimize_algorithm) ();
00082 double (*optimize_estimate_climbing) (unsigned int variable,
00083                                       unsigned int estimate);
00085 double (*optimize_norm) (unsigned int simulation);
00087 Optimize optimize[1];
00088
00092 void
00093 optimize_input (unsigned int simulation,
00094               char *input,
00095               GMappedFile * stencil)
00096 {
00097     char buffer[32], value[32];
00098     GRegex *regex;
00099     FILE *file;
00100     char *buffer2, *buffer3 = NULL, *content;
00101     gsize length;
00102     unsigned int i;
00103
00104     #if DEBUG_OPTIMIZE
00105     fprintf (stderr, "optimize_input: start\n");
00106     #endif
00107
00108     // Checking the file
00109     if (!stencil)
00110         goto optimize_input_end;
00111
00112     // Opening stencil
00113     content = g_mapped_file_get_contents (stencil);
00114     length = g_mapped_file_get_length (stencil);
00115     #if DEBUG_OPTIMIZE
00116     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00117     #endif
00118     file = g_fopen (input, "w");
00119
00120     // Parsing stencil
00121     for (i = 0; i < optimize->nvariables; ++i)
00122     {
00123     #if DEBUG_OPTIMIZE
00124     fprintf (stderr, "optimize_input: variable=%u\n", i);
00125     #endif
00126     snprintf (buffer, 32, "@variable%u@", i + 1);
00127     regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00128                         NULL);
00129     if (i == 0)
00130     {
00131         buffer2 = g_regex_replace_literal (regex, content, length, 0,
00132                                           optimize->label[i],
00133                                           (GRegexMatchFlags) 0, NULL);
00134     #if DEBUG_OPTIMIZE
00135     fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00136     #endif
00137     }
00138     else
00139     {
00140         length = strlen (buffer3);
00141         buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00142                                           optimize->label[i],
00143                                           (GRegexMatchFlags) 0, NULL);
00144         g_free (buffer3);
00145     }
00146     g_regex_unref (regex);
00147     length = strlen (buffer2);
00148     snprintf (buffer, 32, "@value%u@", i + 1);
00149     regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00150                         NULL);
00151     snprintf (value, 32, format[optimize->precision[i]],
00152             optimize->value[simulation * optimize->nvariables + i]);
00153
00154     #if DEBUG_OPTIMIZE
00155     fprintf (stderr, "optimize_input: value=%s\n", value);
00156     #endif
00157     buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00158                                     (GRegexMatchFlags) 0, NULL);
00159     g_free (buffer2);
00160     g_regex_unref (regex);
00161 }
00162
00163 // Saving input file
00164 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00165 g_free (buffer3);
00166 fclose (file);
00167
00168 optimize_input_end:
00169 #if DEBUG_OPTIMIZE
00170     fprintf (stderr, "optimize_input: end\n");

```

```

00171 #endif
00172     return;
00173 }
00174
00181 double
00182 optimize_parse (unsigned int simulation,
00183                unsigned int experiment)
00184 {
00185     unsigned int i;
00186     double e;
00187     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00188         *buffer3, *buffer4;
00189     FILE *file_result;
00190
00191     #if DEBUG_OPTIMIZE
00192     fprintf (stderr, "optimize_parse: start\n");
00193     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00194             simulation, experiment);
00195     #endif
00196
00197     // Opening input files
00198     for (i = 0; i < optimize->ninputs; ++i)
00199     {
00200         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00201     #if DEBUG_OPTIMIZE
00202         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00203     #endif
00204         optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00205     }
00206     for (; i < MAX_NINPUTS; ++i)
00207         strcpy (&input[i][0], "");
00208     #if DEBUG_OPTIMIZE
00209     fprintf (stderr, "optimize_parse: parsing end\n");
00210     #endif
00211
00212     // Performing the simulation
00213     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00214     buffer2 = g_path_get_dirname (optimize->simulator);
00215     buffer3 = g_path_get_basename (optimize->simulator);
00216     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00217     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00218             buffer4, input[0], input[1], input[2], input[3], input[4],
00219             input[5], input[6], input[7], output);
00220     g_free (buffer4);
00221     g_free (buffer3);
00222     g_free (buffer2);
00223     #if DEBUG_OPTIMIZE
00224     fprintf (stderr, "optimize_parse: %s\n", buffer);
00225     #endif
00226     system (buffer);
00227
00228     // Checking the objective value function
00229     if (optimize->evaluator)
00230     {
00231         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00232         buffer2 = g_path_get_dirname (optimize->evaluator);
00233         buffer3 = g_path_get_basename (optimize->evaluator);
00234         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235         snprintf (buffer, 512, "\"%s\" %s %s %s",
00236                 buffer4, output, optimize->experiment[experiment], result);
00237         g_free (buffer4);
00238         g_free (buffer3);
00239         g_free (buffer2);
00240     #if DEBUG_OPTIMIZE
00241         fprintf (stderr, "optimize_parse: %s\n", buffer);
00242         fprintf (stderr, "optimize_parse: result=%s\n", result);
00243     #endif
00244         system (buffer);
00245         file_result = g_fopen (result, "r");
00246         e = atof (fgets (buffer, 512, file_result));
00247         fclose (file_result);
00248     }
00249     else
00250     {
00251     #if DEBUG_OPTIMIZE
00252         fprintf (stderr, "optimize_parse: output=%s\n", output);
00253     #endif
00254         strcpy (result, "");
00255         file_result = g_fopen (output, "r");
00256         e = atof (fgets (buffer, 512, file_result));
00257         fclose (file_result);
00258     }
00259
00260     // Removing files
00261     #if !DEBUG_OPTIMIZE
00262     for (i = 0; i < optimize->ninputs; ++i)
00263     {

```

```

00264         if (optimize->file[i][0])
00265         {
00266             snprintf (buffer, 512, RM " %s", &input[i][0]);
00267             system (buffer);
00268         }
00269     }
00270     snprintf (buffer, 512, RM " %s %s", output, result);
00271     system (buffer);
00272 #endif
00273
00274     // Processing pending events
00275     if (show_pending)
00276         show_pending ();
00277
00278 #if DEBUG_OPTIMIZE
00279     fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282     // Returning the objective function
00283     return e * optimize->weight[experiment];
00284 }
00285
00291 double
00292 optimize_norm_euclidian (unsigned int simulation)
00293 {
00294     double e, ei;
00295     unsigned int i;
00296 #if DEBUG_OPTIMIZE
00297     fprintf (stderr, "optimize_norm_euclidian: start\n");
00298 #endif
00299     e = 0.;
00300     for (i = 0; i < optimize->nexperiments; ++i)
00301     {
00302         ei = optimize_parse (simulation, i);
00303         e += ei * ei;
00304     }
00305     e = sqrt (e);
00306 #if DEBUG_OPTIMIZE
00307     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00308     fprintf (stderr, "optimize_norm_euclidian: end\n");
00309 #endif
00310     return e;
00311 }
00312
00318 double
00319 optimize_norm_maximum (unsigned int simulation)
00320 {
00321     double e, ei;
00322     unsigned int i;
00323 #if DEBUG_OPTIMIZE
00324     fprintf (stderr, "optimize_norm_maximum: start\n");
00325 #endif
00326     e = 0.;
00327     for (i = 0; i < optimize->nexperiments; ++i)
00328     {
00329         ei = fabs (optimize_parse (simulation, i));
00330         e = fmax (e, ei);
00331     }
00332 #if DEBUG_OPTIMIZE
00333     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00334     fprintf (stderr, "optimize_norm_maximum: end\n");
00335 #endif
00336     return e;
00337 }
00338
00344 double
00345 optimize_norm_p (unsigned int simulation)
00346 {
00347     double e, ei;
00348     unsigned int i;
00349 #if DEBUG_OPTIMIZE
00350     fprintf (stderr, "optimize_norm_p: start\n");
00351 #endif
00352     e = 0.;
00353     for (i = 0; i < optimize->nexperiments; ++i)
00354     {
00355         ei = fabs (optimize_parse (simulation, i));
00356         e += pow (ei, optimize->p);
00357     }
00358     e = pow (e, 1. / optimize->p);
00359 #if DEBUG_OPTIMIZE
00360     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00361     fprintf (stderr, "optimize_norm_p: end\n");
00362 #endif
00363     return e;
00364 }
00365

```



```

00371 double
00372 optimize_norm_taxicab (unsigned int simulation)
00373 {
00374     double e;
00375     unsigned int i;
00376     #if DEBUG_OPTIMIZE
00377     fprintf (stderr, "optimize_norm_taxicab: start\n");
00378     #endif
00379     e = 0.;
00380     for (i = 0; i < optimize->nexperiments; ++i)
00381         e += fabs (optimize_parse (simulation, i));
00382     #if DEBUG_OPTIMIZE
00383     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00384     fprintf (stderr, "optimize_norm_taxicab: end\n");
00385     #endif
00386     return e;
00387 }
00388
00392 void
00393 optimize_print ()
00394 {
00395     unsigned int i;
00396     char buffer[512];
00397     #if HAVE_MPI
00398     if (optimize->mpi_rank)
00399         return;
00400     #endif
00401     printf ("%s\n", _("Best result"));
00402     fprintf (optimize->file_result, "%s\n", _("Best result"));
00403     printf ("error = %.15le\n", optimize->error_old[0]);
00404     fprintf (optimize->file_result, "error = %.15le\n", optimize->
00405             error_old[0]);
00406     for (i = 0; i < optimize->nvariables; ++i)
00407     {
00408         snprintf (buffer, 512, "%s = %s\n",
00409                  optimize->label[i], format[optimize->precision[i]]);
00410         printf (buffer, optimize->value_old[i]);
00411         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00412     }
00413     fflush (optimize->file_result);
00414 }
00418 void
00419 optimize_save_variables (unsigned int simulation,
00420                         double error)
00421 {
00422     unsigned int i;
00423     char buffer[64];
00424     #if DEBUG_OPTIMIZE
00425     fprintf (stderr, "optimize_save_variables: start\n");
00426     #endif
00427     for (i = 0; i < optimize->nvariables; ++i)
00428     {
00429         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00430         fprintf (optimize->file_variables, buffer,
00431                 optimize->value[simulation * optimize->nvariables + i]);
00432     }
00433     fprintf (optimize->file_variables, "%.14le\n", error);
00434     fflush (optimize->file_variables);
00435     #if DEBUG_OPTIMIZE
00436     fprintf (stderr, "optimize_save_variables: end\n");
00437     #endif
00438 }
00439
00443 void
00444 optimize_best (unsigned int simulation,
00445               double value)
00446 {
00447     unsigned int i, j;
00448     double e;
00449     #if DEBUG_OPTIMIZE
00450     fprintf (stderr, "optimize_best: start\n");
00451     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00452             optimize->nsaveds, optimize->nbest);
00453     #endif
00454     if (optimize->nsaveds < optimize->nbest
00455         || value < optimize->error_best[optimize->nsaveds - 1])
00456     {
00457         if (optimize->nsaveds < optimize->nbest)
00458             ++optimize->nsaveds;
00459         optimize->error_best[optimize->nsaveds - 1] = value;
00460         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00461         for (i = optimize->nsaveds; --i;)
00462         {
00463             if (optimize->error_best[i] < optimize->error_best[i - 1])
00464             {
00465                 j = optimize->simulation_best[i];

```

```

00466         e = optimize->error_best[i];
00467         optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00468         optimize->error_best[i] = optimize->error_best[i - 1];
00469         optimize->simulation_best[i - 1] = j;
00470         optimize->error_best[i - 1] = e;
00471     }
00472     else
00473         break;
00474 }
00475 }
00476 #if DEBUG_OPTIMIZE
00477 fprintf (stderr, "optimize_best: end\n");
00478 #endif
00479 }
00480
00484 void
00485 optimize_sequential ()
00486 {
00487     unsigned int i;
00488     double e;
00489     #if DEBUG_OPTIMIZE
00490     fprintf (stderr, "optimize_sequential: start\n");
00491     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00492             optimize->nstart, optimize->nend);
00493     #endif
00494     for (i = optimize->nstart; i < optimize->nend; ++i)
00495     {
00496         e = optimize_norm (i);
00497         optimize_best (i, e);
00498         optimize_save_variables (i, e);
00499         if (e < optimize->threshold)
00500         {
00501             optimize->stop = 1;
00502             break;
00503         }
00504     #if DEBUG_OPTIMIZE
00505     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00506     #endif
00507     }
00508     #if DEBUG_OPTIMIZE
00509     fprintf (stderr, "optimize_sequential: end\n");
00510     #endif
00511 }
00512
00518 void *
00519 optimize_thread (ParallelData * data)
00520 {
00521     unsigned int i, thread;
00522     double e;
00523     #if DEBUG_OPTIMIZE
00524     fprintf (stderr, "optimize_thread: start\n");
00525     #endif
00526     thread = data->thread;
00527     #if DEBUG_OPTIMIZE
00528     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00529             optimize->thread[thread], optimize->thread[thread + 1]);
00530     #endif
00531     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00532     {
00533         e = optimize_norm (i);
00534         g_mutex_lock (mutex);
00535         optimize_best (i, e);
00536         optimize_save_variables (i, e);
00537         if (e < optimize->threshold)
00538             optimize->stop = 1;
00539         g_mutex_unlock (mutex);
00540         if (optimize->stop)
00541             break;
00542     #if DEBUG_OPTIMIZE
00543     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00544     #endif
00545     }
00546     #if DEBUG_OPTIMIZE
00547     fprintf (stderr, "optimize_thread: end\n");
00548     #endif
00549     g_thread_exit (NULL);
00550     return NULL;
00551 }
00552
00556 void
00557 optimize_merge (unsigned int nsaveds,
00558                unsigned int *simulation_best,
00559                double *error_best)
00560 {
00561     unsigned int i, j, k, s[optimize->nbest];
00562     double e[optimize->nbest];

```

```

00565 #if DEBUG_OPTIMIZE
00566     fprintf (stderr, "optimize_merge: start\n");
00567 #endif
00568     i = j = k = 0;
00569     do
00570     {
00571         if (i == optimize->nsaveds)
00572         {
00573             s[k] = simulation_best[j];
00574             e[k] = error_best[j];
00575             ++j;
00576             ++k;
00577             if (j == nsaveds)
00578                 break;
00579         }
00580         else if (j == nsaveds)
00581         {
00582             s[k] = optimize->simulation_best[i];
00583             e[k] = optimize->error_best[i];
00584             ++i;
00585             ++k;
00586             if (i == optimize->nsaveds)
00587                 break;
00588         }
00589         else if (optimize->error_best[i] > error_best[j])
00590         {
00591             s[k] = simulation_best[j];
00592             e[k] = error_best[j];
00593             ++j;
00594             ++k;
00595         }
00596         else
00597         {
00598             s[k] = optimize->simulation_best[i];
00599             e[k] = optimize->error_best[i];
00600             ++i;
00601             ++k;
00602         }
00603     }
00604     while (k < optimize->nbest);
00605     optimize->nsaveds = k;
00606     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00607     memcpy (optimize->error_best, e, k * sizeof (double));
00608 #if DEBUG_OPTIMIZE
00609     fprintf (stderr, "optimize_merge: end\n");
00610 #endif
00611 }
00612
00616 #if HAVE_MPI
00617 void
00618 optimize_synchronise ()
00619 {
00620     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00621     double error_best[optimize->nbest];
00622     MPI_Status mpi_stat;
00623 #if DEBUG_OPTIMIZE
00624     fprintf (stderr, "optimize_synchronise: start\n");
00625 #endif
00626     if (optimize->mpi_rank == 0)
00627     {
00628         for (i = 1; i < ntasks; ++i)
00629         {
00630             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00631             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00632                      MPI_COMM_WORLD, &mpi_stat);
00633             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00634                      MPI_COMM_WORLD, &mpi_stat);
00635             optimize_merge (nsaveds, simulation_best, error_best);
00636             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637             if (stop)
00638                 optimize->stop = 1;
00639         }
00640         for (i = 1; i < ntasks; ++i)
00641             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00642     }
00643     else
00644     {
00645         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00646         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00647                  MPI_COMM_WORLD);
00648         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00649                  MPI_COMM_WORLD);
00650         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00651         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00652         if (stop)
00653             optimize->stop = 1;
00654     }

```

```

00655 #if DEBUG_OPTIMIZE
00656     fprintf (stderr, "optimize_synchronise: end\n");
00657 #endif
00658 }
00659 #endif
00660
00664 void
00665 optimize_sweep ()
00666 {
00667     unsigned int i, j, k, l;
00668     double e;
00669     GThread *thread[nthreads];
00670     ParallelData data[nthreads];
00671 #if DEBUG_OPTIMIZE
00672     fprintf (stderr, "optimize_sweep: start\n");
00673 #endif
00674     for (i = 0; i < optimize->nsimulations; ++i)
00675     {
00676         k = i;
00677         for (j = 0; j < optimize->nvariables; ++j)
00678         {
00679             l = k % optimize->nsweeps[j];
00680             k /= optimize->nsweeps[j];
00681             e = optimize->rangemin[j];
00682             if (optimize->nsweeps[j] > 1)
00683                 e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00684                     / (optimize->nsweeps[j] - 1);
00685             optimize->value[i * optimize->nvariables + j] = e;
00686         }
00687     }
00688     optimize->nsaveds = 0;
00689     if (nthreads <= 1)
00690         optimize_sequential ();
00691     else
00692     {
00693         for (i = 0; i < nthreads; ++i)
00694         {
00695             data[i].thread = i;
00696             thread[i]
00697                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00698         }
00699         for (i = 0; i < nthreads; ++i)
00700             g_thread_join (thread[i]);
00701     }
00702 #if HAVE_MPI
00703     // Communicating tasks results
00704     optimize_synchronise ();
00705 #endif
00706 #if DEBUG_OPTIMIZE
00707     fprintf (stderr, "optimize_sweep: end\n");
00708 #endif
00709 }
00710
00714 void
00715 optimize_MonteCarlo ()
00716 {
00717     unsigned int i, j;
00718     GThread *thread[nthreads];
00719     ParallelData data[nthreads];
00720 #if DEBUG_OPTIMIZE
00721     fprintf (stderr, "optimize_MonteCarlo: start\n");
00722 #endif
00723     for (i = 0; i < optimize->nsimulations; ++i)
00724     {
00725         for (j = 0; j < optimize->nvariables; ++j)
00726             optimize->value[i * optimize->nvariables + j]
00727                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00728                     * (optimize->rangemax[j] - optimize->rangemin[j]);
00729     }
00730     optimize->nsaveds = 0;
00731     if (nthreads <= 1)
00732         optimize_sequential ();
00733     else
00734     {
00735         for (i = 0; i < nthreads; ++i)
00736         {
00737             data[i].thread = i;
00738             thread[i]
00739                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00740         }
00741         for (i = 0; i < nthreads; ++i)
00742             g_thread_join (thread[i]);
00743     }
00744 #if HAVE_MPI
00745     // Communicating tasks results
00746     optimize_synchronise ();
00747 #endif
00748 #if DEBUG_OPTIMIZE
00749     fprintf (stderr, "optimize_MonteCarlo: end\n");
00750 #endif

```

```

00748 #endif
00749 }
00750
00754 void
00755 optimize_orthogonal ()
00756 {
00757     unsigned int i, j, k, l;
00758     double e;
00759     GThread *thread[nthreads];
00760     ParallelData data[nthreads];
00761     #if DEBUG_OPTIMIZE
00762     fprintf (stderr, "optimize_orthogonal: start\n");
00763     #endif
00764     for (i = 0; i < optimize->nsimulations; ++i)
00765     {
00766         k = i;
00767         for (j = 0; j < optimize->nvariables; ++j)
00768         {
00769             l = k % optimize->nsweeps[j];
00770             k /= optimize->nsweeps[j];
00771             e = optimize->rangemin[j];
00772             if (optimize->nsweeps[j] > 1)
00773                 e += (l + gsl_rng_uniform (optimize->rng))
00774                     * (optimize->rangemax[j] - optimize->rangemin[j])
00775                     / optimize->nsweeps[j];
00776             optimize->value[i * optimize->nvariables + j] = e;
00777         }
00778     }
00779     optimize->nsaveds = 0;
00780     if (nthreads <= 1)
00781         optimize_sequential ();
00782     else
00783     {
00784         for (i = 0; i < nthreads; ++i)
00785         {
00786             data[i].thread = i;
00787             thread[i]
00788                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00789         }
00790         for (i = 0; i < nthreads; ++i)
00791             g_thread_join (thread[i]);
00792     }
00793     #if HAVE_MPI
00794     // Communicating tasks results
00795     optimize_synchronise ();
00796     #endif
00797     #if DEBUG_OPTIMIZE
00798     fprintf (stderr, "optimize_orthogonal: end\n");
00799     #endif
00800 }
00801
00805 void
00806 optimize_best_climbing (unsigned int simulation,
00807                         double value)
00808 {
00809     #if DEBUG_OPTIMIZE
00810     fprintf (stderr, "optimize_best_climbing: start\n");
00811     fprintf (stderr,
00812             "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00813             simulation, value, optimize->error_best[0]);
00814     #endif
00815     if (value < optimize->error_best[0])
00816     {
00817         optimize->error_best[0] = value;
00818         optimize->simulation_best[0] = simulation;
00819     }
00820     #if DEBUG_OPTIMIZE
00821     fprintf (stderr,
00822             "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00823             simulation, value);
00824     #endif
00825     #if DEBUG_OPTIMIZE
00826     fprintf (stderr, "optimize_best_climbing: end\n");
00827     #endif
00828 }
00829
00833 void
00834 optimize_climbing_sequential (unsigned int simulation)
00835 {
00836     double e;
00837     unsigned int i, j;
00838     #if DEBUG_OPTIMIZE
00839     fprintf (stderr, "optimize_climbing_sequential: start\n");
00840     fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00841             "nend_climbing=%u\n",
00842             optimize->nstart_climbing, optimize->nend_climbing);
00843     #endif

```

```

00844     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00845     {
00846         j = simulation + i;
00847         e = optimize_norm (j);
00848         optimize_best_climbing (j, e);
00849         optimize_save_variables (j, e);
00850         if (e < optimize->threshold)
00851         {
00852             optimize->stop = 1;
00853             break;
00854         }
00855 #if DEBUG_OPTIMIZE
00856     fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
00857 #endif
00858     }
00859 #if DEBUG_OPTIMIZE
00860     fprintf (stderr, "optimize_climbing_sequential: end\n");
00861 #endif
00862 }
00863
00864 void *
00870 optimize_climbing_thread (ParallelData * data)
00871 {
00872     unsigned int i, thread;
00873     double e;
00874 #if DEBUG_OPTIMIZE
00875     fprintf (stderr, "optimize_climbing_thread: start\n");
00876 #endif
00877     thread = data->thread;
00878 #if DEBUG_OPTIMIZE
00879     fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00880             thread,
00881             optimize->thread_climbing[thread],
00882             optimize->thread_climbing[thread + 1]);
00883 #endif
00884     for (i = optimize->thread_climbing[thread];
00885          i < optimize->thread_climbing[thread + 1]; ++i)
00886     {
00887         e = optimize_norm (i);
00888         g_mutex_lock (mutex);
00889         optimize_best_climbing (i, e);
00890         optimize_save_variables (i, e);
00891         if (e < optimize->threshold)
00892         {
00893             optimize->stop = 1;
00894             g_mutex_unlock (mutex);
00895             if (optimize->stop)
00896                 break;
00897 #if DEBUG_OPTIMIZE
00898             fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00899 #endif
00900 #if DEBUG_OPTIMIZE
00901             fprintf (stderr, "optimize_climbing_thread: end\n");
00902 #endif
00903         g_thread_exit (NULL);
00904         return NULL;
00905     }
00906
00907 double
00911 optimize_estimate_climbing_random (unsigned int variable,
00912                                   unsigned int estimate
00913                                   __attribute__ ((unused)))
00914 {
00915     double x;
00916 #if DEBUG_OPTIMIZE
00917     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00918 #endif
00919     x = optimize->climbing[variable]
00920         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00921 #if DEBUG_OPTIMIZE
00922     fprintf (stderr, "optimize_estimate_climbing_random: climbing=%u=%lg\n",
00923             variable, x);
00924     fprintf (stderr, "optimize_estimate_climbing_random: end\n");
00925 #endif
00926     return x;
00927 }
00928
00929 double
00933 optimize_estimate_climbing_coordinates (unsigned int variable,
00934                                         unsigned int estimate)
00935 {
00936     double x;
00937 #if DEBUG_OPTIMIZE
00938     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00939 #endif
00940     x = optimize->climbing[variable];
00941     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))

```

```

00946     {
00947         if (estimate & 1)
00948             x += optimize->step[variable];
00949         else
00950             x -= optimize->step[variable];
00951     }
00952 #if DEBUG_OPTIMIZE
00953     fprintf (stderr,
00954             "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
00955             variable, x);
00956     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00957 #endif
00958     return x;
00959 }
00960
00961 void
00962 optimize_step_climbing (unsigned int simulation)
00963 {
00964     GThread *thread[nthreads_climbing];
00965     ParallelData data[nthreads_climbing];
00966     unsigned int i, j, k, b;
00967 #if DEBUG_OPTIMIZE
00968     fprintf (stderr, "optimize_step_climbing: start\n");
00969 #endif
00970     for (i = 0; i < optimize->nestimates; ++i)
00971     {
00972         k = (simulation + i) * optimize->nvariables;
00973         b = optimize->simulation_best[0] * optimize->nvariables;
00974 #if DEBUG_OPTIMIZE
00975         fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
00976                 simulation + i, optimize->simulation_best[0]);
00977 #endif
00978         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00979         {
00980             #if DEBUG_OPTIMIZE
00981             fprintf (stderr,
00982                     "optimize_step_climbing: estimate=%u best%u=%.14le\n",
00983                     i, j, optimize->value[b]);
00984             #endif
00985             optimize->value[k]
00986                 = optimize->value[b] + optimize_estimate_climbing (j, i);
00987             optimize->value[k] = fmin (fmax (optimize->value[k],
00988                                             optimize->rangeminabs[j]),
00989                                     optimize->rangemaxabs[j]);
00990 #if DEBUG_OPTIMIZE
00991             fprintf (stderr,
00992                     "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
00993                     i, j, optimize->value[k]);
00994             #endif
00995         }
00996     }
00997     if (nthreads_climbing == 1)
00998         optimize_climbing_sequential (simulation);
00999     else
01000     {
01001         for (i = 0; i <= nthreads_climbing; ++i)
01002         {
01003             optimize->thread_climbing[i]
01004                 = simulation + optimize->nstart_climbing
01005                 + i * (optimize->nend_climbing - optimize->
01006                     nstart_climbing)
01007                 / nthreads_climbing;
01008 #if DEBUG_OPTIMIZE
01009             fprintf (stderr,
01010                     "optimize_step_climbing: i=%u thread_climbing=%u\n",
01011                     i, optimize->thread_climbing[i]);
01012 #endif
01013         }
01014         for (i = 0; i < nthreads_climbing; ++i)
01015         {
01016             data[i].thread = i;
01017             thread[i] = g_thread_new
01018                 (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01019         }
01020         for (i = 0; i < nthreads_climbing; ++i)
01021             g_thread_join (thread[i]);
01022     }
01023 #if DEBUG_OPTIMIZE
01024     fprintf (stderr, "optimize_step_climbing: end\n");
01025 #endif
01026 }
01027 void
01028 optimize_climbing ()
01029 {
01030     unsigned int i, j, k, b, s, adjust;
01031 #if DEBUG_OPTIMIZE

```

```

01038     fprintf (stderr, "optimize_climbing: start\n");
01039 #endif
01040     for (i = 0; i < optimize->nvariables; ++i)
01041         optimize->climbing[i] = 0.;
01042     b = optimize->simulation_best[0] * optimize->nvariables;
01043     s = optimize->nsimulations;
01044     adjust = 1;
01045     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01046     {
01047 #if DEBUG_OPTIMIZE
01048         fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01049                 i, optimize->simulation_best[0]);
01050 #endif
01051         optimize_step_climbing (s);
01052         k = optimize->simulation_best[0] * optimize->nvariables;
01053 #if DEBUG_OPTIMIZE
01054         fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01055                 i, optimize->simulation_best[0]);
01056 #endif
01057         if (k == b)
01058         {
01059             if (adjust)
01060                 for (j = 0; j < optimize->nvariables; ++j)
01061                     optimize->step[j] *= 0.5;
01062             for (j = 0; j < optimize->nvariables; ++j)
01063                 optimize->climbing[j] = 0.;
01064             adjust = 1;
01065         }
01066         else
01067         {
01068             for (j = 0; j < optimize->nvariables; ++j)
01069             {
01070 #if DEBUG_OPTIMIZE
01071                 fprintf (stderr,
01072                         "optimize_climbing: best%u=%%.14le old%u=%%.14le\n",
01073                         j, optimize->value[k + j], j, optimize->value[b + j]);
01074 #endif
01075                 optimize->climbing[j]
01076                     = (1. - optimize->relaxation) * optimize->climbing[j]
01077                       + optimize->relaxation
01078                       * (optimize->value[k + j] - optimize->value[b + j]);
01079 #if DEBUG_OPTIMIZE
01080                 fprintf (stderr, "optimize_climbing: climbing%u=%%.14le\n",
01081                         j, optimize->climbing[j]);
01082 #endif
01083             }
01084             adjust = 0;
01085         }
01086     }
01087 #if DEBUG_OPTIMIZE
01088     fprintf (stderr, "optimize_climbing: end\n");
01089 #endif
01090 }
01091
01092 double
01093 optimize_genetic_objective (Entity * entity)
01094 {
01095     unsigned int j;
01096     double objective;
01097     char buffer[64];
01098 #if DEBUG_OPTIMIZE
01099     fprintf (stderr, "optimize_genetic_objective: start\n");
01100 #endif
01101     for (j = 0; j < optimize->nvariables; ++j)
01102     {
01103         optimize->value[entity->id * optimize->nvariables + j]
01104             = genetic_get_variable (entity, optimize->genetic_variable + j);
01105     }
01106     objective = optimize_norm (entity->id);
01107     g_mutex_lock (mutex);
01108     for (j = 0; j < optimize->nvariables; ++j)
01109     {
01110         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01111         fprintf (optimize->file_variables, buffer,
01112                 genetic_get_variable (entity, optimize->genetic_variable + j));
01113     }
01114     fprintf (optimize->file_variables, "%.14le\n", objective);
01115     g_mutex_unlock (mutex);
01116 #if DEBUG_OPTIMIZE
01117     fprintf (stderr, "optimize_genetic_objective: end\n");
01118 #endif
01119     return objective;
01120 }
01121
01122 void
01123 optimize_genetic ()
01124 {

```



```

01133     double *best_variable = NULL;
01134     char *best_genome = NULL;
01135     double best_objective = 0.;
01136     #if DEBUG_OPTIMIZE
01137     fprintf (stderr, "optimize_genetic: start\n");
01138     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01139             nthreads);
01140     fprintf (stderr,
01141             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01142             optimize->nvariables, optimize->nsimulations, optimize->
niterations);
01143     fprintf (stderr,
01144             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01145             optimize->mutation_ratio, optimize->reproduction_ratio,
01146             optimize->adaptation_ratio);
01147     #endif
01148     genetic_algorithm_default (optimize->nvariables,
01149                               optimize->genetic_variable,
01150                               optimize->nsimulations,
01151                               optimize->niterations,
01152                               optimize->mutation_ratio,
01153                               optimize->reproduction_ratio,
01154                               optimize->adaptation_ratio,
01155                               optimize->seed,
01156                               optimize->threshold,
01157                               &optimize_genetic_objective,
01158                               &best_genome, &best_variable, &best_objective);
01159     #if DEBUG_OPTIMIZE
01160     fprintf (stderr, "optimize_genetic: the best\n");
01161     #endif
01162     optimize->error_old = (double *) g_malloc (sizeof (double));
01163     optimize->value_old
01164         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01165     optimize->error_old[0] = best_objective;
01166     memcpy (optimize->value_old, best_variable,
01167            optimize->nvariables * sizeof (double));
01168     g_free (best_genome);
01169     g_free (best_variable);
01170     optimize_print ();
01171     #if DEBUG_OPTIMIZE
01172     fprintf (stderr, "optimize_genetic: end\n");
01173     #endif
01174 }
01175
01176 void
01177 optimize_save_old ()
01178 {
01179     unsigned int i, j;
01180     #if DEBUG_OPTIMIZE
01181     fprintf (stderr, "optimize_save_old: start\n");
01182     #endif
01183     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01184     #endif
01185     memcpy (optimize->error_old, optimize->error_best,
01186            optimize->nbest * sizeof (double));
01187     for (i = 0; i < optimize->nbest; ++i)
01188     {
01189         j = optimize->simulation_best[i];
01190         #if DEBUG_OPTIMIZE
01191         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01192         #endif
01193         #endif
01194         memcpy (optimize->value_old + i * optimize->nvariables,
01195                optimize->value + j * optimize->nvariables,
01196                optimize->nvariables * sizeof (double));
01197     }
01198     #if DEBUG_OPTIMIZE
01199     for (i = 0; i < optimize->nvariables; ++i)
01200         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01201                 i, optimize->value_old[i]);
01202     fprintf (stderr, "optimize_save_old: end\n");
01203     #endif
01204 }
01205
01206 void
01207 optimize_merge_old ()
01208 {
01209     unsigned int i, j, k;
01210     double v[optimize->nbest * optimize->nvariables], e[optimize->
nbest],
01211            *enew, *eold;
01212     #if DEBUG_OPTIMIZE
01213     fprintf (stderr, "optimize_merge_old: start\n");
01214     #endif
01215     anew = optimize->error_best;
01216     eold = optimize->error_old;
01217     i = j = k = 0;
01218     do
01219     {

```

```

01225     if (*enew < *eold)
01226     {
01227         memcpy (v + k * optimize->nvariables,
01228             optimize->value
01229             + optimize->simulation_best[i] * optimize->
01230             nvariables,
01231             optimize->nvariables * sizeof (double));
01232         e[k] = *enew;
01233         ++k;
01234         ++enew;
01235         ++i;
01236     }
01237     else
01238     {
01239         memcpy (v + k * optimize->nvariables,
01240             optimize->value_old + j * optimize->nvariables,
01241             optimize->nvariables * sizeof (double));
01242         e[k] = *eold;
01243         ++k;
01244         ++eold;
01245         ++j;
01246     }
01247     while (k < optimize->nbest);
01248     memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01249     memcpy (optimize->error_old, e, k * sizeof (double));
01250     #if DEBUG_OPTIMIZE
01251     fprintf (stderr, "optimize_merge_old: end\n");
01252     #endif
01253 }
01254
01255 void
01260 optimize_refine ()
01261 {
01262     unsigned int i, j;
01263     double d;
01264     #if HAVE_MPI
01265     MPI_Status mpi_stat;
01266     #endif
01267     #if DEBUG_OPTIMIZE
01268     fprintf (stderr, "optimize_refine: start\n");
01269     #endif
01270     #if HAVE_MPI
01271     if (!optimize->mpi_rank)
01272     {
01273     #endif
01274         for (j = 0; j < optimize->nvariables; ++j)
01275         {
01276             optimize->rangemin[j] = optimize->rangemax[j]
01277             = optimize->value_old[j];
01278         }
01279         for (i = 0; ++i < optimize->nbest;)
01280         {
01281             for (j = 0; j < optimize->nvariables; ++j)
01282             {
01283                 optimize->rangemin[j]
01284                 = fmin (optimize->rangemin[j],
01285                     optimize->value_old[i * optimize->nvariables + j]);
01286                 optimize->rangemax[j]
01287                 = fmax (optimize->rangemax[j],
01288                     optimize->value_old[i * optimize->nvariables + j]);
01289             }
01290         }
01291         for (j = 0; j < optimize->nvariables; ++j)
01292         {
01293             d = optimize->tolerance
01294             * (optimize->rangemax[j] - optimize->rangemin[j]);
01295             switch (optimize->algorithm)
01296             {
01297             case ALGORITHM_MONTE_CARLO:
01298                 d *= 0.5;
01299                 break;
01300             default:
01301                 if (optimize->nsweeps[j] > 1)
01302                     d /= optimize->nsweeps[j] - 1;
01303                 else
01304                     d = 0.;
01305             }
01306             optimize->rangemin[j] -= d;
01307             optimize->rangemin[j]
01308             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01309             optimize->rangemax[j] += d;
01310             optimize->rangemax[j]
01311             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01312             printf ("%s min=%lg max=%lg\n", optimize->label[j],
01313                 optimize->rangemin[j], optimize->rangemax[j]);
01314             fprintf (optimize->file_result, "%s min=%lg max=%lg\n",

```

```

01315             optimize->label[j], optimize->rangemin[j],
01316             optimize->rangemax[j]);
01317         }
01318 #if HAVE_MPI
01319     for (i = 1; i < ntasks; ++i)
01320     {
01321         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01322                 1, MPI_COMM_WORLD);
01323         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01324                 1, MPI_COMM_WORLD);
01325     }
01326 }
01327 else
01328 {
01329     MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01330             MPI_COMM_WORLD, &mpi_stat);
01331     MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01332             MPI_COMM_WORLD, &mpi_stat);
01333 }
01334 #endif
01335 #if DEBUG_OPTIMIZE
01336 fprintf (stderr, "optimize_refine: end\n");
01337 #endif
01338 }
01339
01340 void
01341 optimize_step ()
01342 {
01343     #if DEBUG_OPTIMIZE
01344     fprintf (stderr, "optimize_step: start\n");
01345     #endif
01346     optimize_algorithm ();
01347     if (optimize->nsteps)
01348         optimize_climbing ();
01349     #if DEBUG_OPTIMIZE
01350     fprintf (stderr, "optimize_step: end\n");
01351     #endif
01352 }
01353
01354 void
01355 optimize_iterate ()
01356 {
01357     unsigned int i;
01358     #if DEBUG_OPTIMIZE
01359     fprintf (stderr, "optimize_iterate: start\n");
01360     #endif
01361     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01362     optimize->value_old =
01363         (double *) g_malloc (optimize->nbest * optimize->nvariables *
01364                             sizeof (double));
01365     optimize_step ();
01366     optimize_save_old ();
01367     optimize_refine ();
01368     optimize_print ();
01369     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01370     {
01371         optimize_step ();
01372         optimize_merge_old ();
01373         optimize_refine ();
01374         optimize_print ();
01375     }
01376     #if DEBUG_OPTIMIZE
01377     fprintf (stderr, "optimize_iterate: end\n");
01378     #endif
01379 }
01380
01381 void
01382 optimize_free ()
01383 {
01384     unsigned int i, j;
01385     #if DEBUG_OPTIMIZE
01386     fprintf (stderr, "optimize_free: start\n");
01387     #endif
01388     for (j = 0; j < optimize->ninputs; ++j)
01389     {
01390         for (i = 0; i < optimize->nexperiments; ++i)
01391             g_mapped_file_unref (optimize->file[j][i]);
01392         g_free (optimize->file[j]);
01393     }
01394     g_free (optimize->error_old);
01395     g_free (optimize->value_old);
01396     g_free (optimize->value);
01397     g_free (optimize->genetic_variable);
01398     #if DEBUG_OPTIMIZE
01399     fprintf (stderr, "optimize_free: end\n");
01400     #endif
01401 }

```

```

01411
01415 void
01416 optimize_open ()
01417 {
01418     GTimeZone *tz;
01419     GDateTime *t0, *t;
01420     unsigned int i, j;
01421
01422     #if DEBUG_OPTIMIZE
01423         char *buffer;
01424         fprintf (stderr, "optimize_open: start\n");
01425     #endif
01426
01427     // Getting initial time
01428     #if DEBUG_OPTIMIZE
01429         fprintf (stderr, "optimize_open: getting initial time\n");
01430     #endif
01431     tz = g_time_zone_new_utc ();
01432     t0 = g_date_time_new_now (tz);
01433
01434     // Obtaining and initing the pseudo-random numbers generator seed
01435     #if DEBUG_OPTIMIZE
01436         fprintf (stderr, "optimize_open: getting initial seed\n");
01437     #endif
01438     if (optimize->seed == DEFAULT_RANDOM_SEED)
01439         optimize->seed = input->seed;
01440     gsl_rng_set (optimize->rng, optimize->seed);
01441
01442     // Replacing the working directory
01443     #if DEBUG_OPTIMIZE
01444         fprintf (stderr, "optimize_open: replacing the working directory\n");
01445     #endif
01446     g_chdir (input->directory);
01447
01448     // Getting results file names
01449     optimize->result = input->result;
01450     optimize->variables = input->variables;
01451
01452     // Obtaining the simulator file
01453     optimize->simulator = input->simulator;
01454
01455     // Obtaining the evaluator file
01456     optimize->evaluator = input->evaluator;
01457
01458     // Reading the algorithm
01459     optimize->algorithm = input->algorithm;
01460     switch (optimize->algorithm)
01461     {
01462         case ALGORITHM_MONTE_CARLO:
01463             optimize_algorithm = optimize_MonteCarlo;
01464             break;
01465         case ALGORITHM_SWEEP:
01466             optimize_algorithm = optimize_sweep;
01467             break;
01468         case ALGORITHM_ORTHOGONAL:
01469             optimize_algorithm = optimize_orthogonal;
01470             break;
01471         default:
01472             optimize_algorithm = optimize_genetic;
01473             optimize->mutation_ratio = input->mutation_ratio;
01474             optimize->reproduction_ratio = input->
01475             reproduction_ratio;
01476             optimize->adaptation_ratio = input->adaptation_ratio;
01477     }
01478     optimize->nvariables = input->nvariables;
01479     optimize->nsimulations = input->nsimulations;
01480     optimize->niterations = input->niterations;
01481     optimize->nbest = input->nbest;
01482     optimize->tolerance = input->tolerance;
01483     optimize->nsteps = input->nsteps;
01484     optimize->nestimates = 0;
01485     optimize->threshold = input->threshold;
01486     optimize->stop = 0;
01487     if (input->nsteps)
01488     {
01489         optimize->relaxation = input->relaxation;
01490         switch (input->climbing)
01491         {
01492             case CLIMBING_METHOD_COORDINATES:
01493                 optimize->nestimates = 2 * optimize->nvariables;
01494                 optimize_estimate_climbing =
01495                 optimize_estimate_climbing_coordinates;
01496                 break;
01497             default:
01498                 optimize->nestimates = input->nestimates;
01499                 optimize_estimate_climbing =
01500                 optimize_estimate_climbing_random;

```

```

01498     }
01499 }
01500
01501 #if DEBUG_OPTIMIZE
01502 fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01503 #endif
01504 optimize->simulation_best
01505     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01506 optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01507
01508 // Reading the experimental data
01509 #if DEBUG_OPTIMIZE
01510 buffer = g_get_current_dir ();
01511 fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01512 g_free (buffer);
01513 #endif
01514 optimize->nexperiments = input->nexperiments;
01515 optimize->ninputs = input->experiment->ninputs;
01516 optimize->experiment
01517     = (char **) alloca (input->nexperiments * sizeof (char *));
01518 optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01519 for (i = 0; i < input->experiment->ninputs; ++i)
01520     optimize->file[i] = (GMappedFile **)
01521         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01522 for (i = 0; i < input->nexperiments; ++i)
01523     {
01524 #if DEBUG_OPTIMIZE
01525         fprintf (stderr, "optimize_open: i=%u\n", i);
01526 #endif
01527         optimize->experiment[i] = input->experiment[i].
01528             name;
01529         optimize->weight[i] = input->experiment[i].weight;
01530 #if DEBUG_OPTIMIZE
01531         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01532             optimize->experiment[i], optimize->weight[i]);
01533 #endif
01534         for (j = 0; j < input->experiment->ninputs; ++j)
01535             {
01536 #if DEBUG_OPTIMIZE
01537                 fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01538 #endif
01539                 optimize->file[j][i]
01540                     = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01541             }
01542     }
01543 // Reading the variables data
01544 #if DEBUG_OPTIMIZE
01545 fprintf (stderr, "optimize_open: reading variables\n");
01546 #endif
01547 optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01548 j = input->nvariables * sizeof (double);
01549 optimize->rangemin = (double *) alloca (j);
01550 optimize->rangeminabs = (double *) alloca (j);
01551 optimize->rangemax = (double *) alloca (j);
01552 optimize->rangemaxabs = (double *) alloca (j);
01553 optimize->step = (double *) alloca (j);
01554 j = input->nvariables * sizeof (unsigned int);
01555 optimize->precision = (unsigned int *) alloca (j);
01556 optimize->nsweeps = (unsigned int *) alloca (j);
01557 optimize->nbits = (unsigned int *) alloca (j);
01558 for (i = 0; i < input->nvariables; ++i)
01559     {
01560         optimize->label[i] = input->variable[i].name;
01561         optimize->rangemin[i] = input->variable[i].rangemin;
01562         optimize->rangeminabs[i] = input->variable[i].
01563             rangeminabs;
01564         optimize->rangemax[i] = input->variable[i].rangemax;
01565         optimize->rangemaxabs[i] = input->variable[i].
01566             rangemaxabs;
01567         optimize->precision[i] = input->variable[i].
01568             precision;
01569         optimize->step[i] = input->variable[i].step;
01570         optimize->nsweeps[i] = input->variable[i].nsweeps;
01571         optimize->nbits[i] = input->variable[i].nbits;
01572     }
01573 if (input->algorithm == ALGORITHM_SWEEP
01574     || input->algorithm == ALGORITHM_ORTHOGONAL)
01575     {
01576         optimize->nsimulations = 1;
01577         for (i = 0; i < input->nvariables; ++i)
01578             {
01579                 optimize->nsimulations *= optimize->nsweeps[i];
01580 #if DEBUG_OPTIMIZE
01581                 fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01582                     optimize->nsweeps[i], optimize->nsimulations);
01583 #endif
01584             }
01585     }

```

```

01581     }
01582 }
01583 if (optimize->nsteps)
01584     optimize->climbing
01585     = (double *) alloca (optimize->nvariables * sizeof (double));
01586
01587 // Setting error norm
01588 switch (input->norm)
01589 {
01590     case ERROR_NORM_EUCLIDIAN:
01591         optimize_norm = optimize_norm_euclidian;
01592         break;
01593     case ERROR_NORM_MAXIMUM:
01594         optimize_norm = optimize_norm_maximum;
01595         break;
01596     case ERROR_NORM_P:
01597         optimize_norm = optimize_norm_p;
01598         optimize->p = input->p;
01599         break;
01600     default:
01601         optimize_norm = optimize_norm_taxicab;
01602 }
01603
01604 // Allocating values
01605 #if DEBUG_OPTIMIZE
01606 fprintf (stderr, "optimize_open: allocating variables\n");
01607 fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01608         optimize->nvariables, optimize->algorithm);
01609 #endif
01610 optimize->genetic_variable = NULL;
01611 if (optimize->algorithm == ALGORITHM_GENETIC)
01612 {
01613     optimize->genetic_variable = (GeneticVariable *)
01614     g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01615     for (i = 0; i < optimize->nvariables; ++i)
01616     {
01617         #if DEBUG_OPTIMIZE
01618             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01619                     i, optimize->rangemin[i], optimize->rangemax[i],
01620                     optimize->nbits[i]);
01621         #endif
01622         optimize->genetic_variable[i].minimum = optimize->
01623         rangemin[i];
01624         optimize->genetic_variable[i].maximum = optimize->
01625         rangemax[i];
01626         optimize->genetic_variable[i].nbits = optimize->nbits[i];
01627     }
01628 #if DEBUG_OPTIMIZE
01629     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01630             optimize->nvariables, optimize->nsimulations);
01631 #endif
01632 optimize->value = (double *)
01633 g_malloc ((optimize->nsimulations
01634           + optimize->nestimates * optimize->nsteps)
01635           * optimize->nvariables * sizeof (double));
01636 // Calculating simulations to perform for each task
01637 #if HAVE_MPI
01638 #if DEBUG_OPTIMIZE
01639     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01640             optimize->mpi_rank, ntasks);
01641 #endif
01642     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01643     ntasks;
01644     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01645     ntasks;
01646     if (optimize->nsteps)
01647     {
01648         optimize->nstart_climbing
01649         = optimize->mpi_rank * optimize->nestimates / ntasks;
01650         optimize->nend_climbing
01651         = (1 + optimize->mpi_rank) * optimize->nestimates /
01652         ntasks;
01653     }
01654 #else
01655     optimize->nstart = 0;
01656     optimize->nend = optimize->nsimulations;
01657     if (optimize->nsteps)
01658     {
01659         optimize->nstart_climbing = 0;
01660         optimize->nend_climbing = optimize->nestimates;
01661     }
01662 #endif
01663 #if DEBUG_OPTIMIZE
01664     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01665             optimize->nend);

```

```

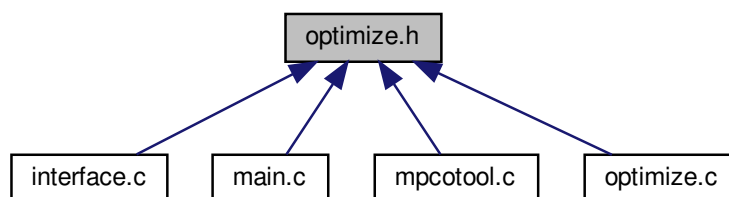
01663 #endif
01664
01665 // Calculating simulations to perform for each thread
01666 optimize->thread
01667 = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01668 for (i = 0; i <= nthreads; ++i)
01669 {
01670     optimize->thread[i] = optimize->nstart
01671         + i * (optimize->end - optimize->nstart) / nthreads;
01672 #if DEBUG_OPTIMIZE
01673     fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01674             optimize->thread[i]);
01675 #endif
01676 }
01677 if (optimize->nsteps)
01678     optimize->thread_climbing = (unsigned int *)
01679         alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01680
01681 // Opening result files
01682 optimize->file_result = g_fopen (optimize->result, "w");
01683 optimize->file_variables = g_fopen (optimize->variables, "w");
01684
01685 // Performing the algorithm
01686 switch (optimize->algorithm)
01687 {
01688     // Genetic algorithm
01689     case ALGORITHM_GENETIC:
01690         optimize_genetic ();
01691         break;
01692
01693     // Iterative algorithm
01694     default:
01695         optimize_iterate ();
01696 }
01697
01698 // Getting calculation time
01699 t = g_date_time_new_now (tz);
01700 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01701 g_date_time_unref (t);
01702 g_date_time_unref (t0);
01703 g_time_zone_unref (tz);
01704 printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01705 fprintf (optimize->file_result, "%s = %.6lg s\n",
01706         _("Calculation time"), optimize->calculation_time);
01707
01708 // Closing result files
01709 fclose (optimize->file_variables);
01710 fclose (optimize->file_result);
01711
01712 #if DEBUG_OPTIMIZE
01713     fprintf (stderr, "optimize_open: end\n");
01714 #endif
01715 }

```

4.21 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_input](#) (unsigned int simulation, char *[input](#), GMappedFile *[stencil](#))
- double [optimize_parse](#) (unsigned int simulation, unsigned int experiment)
- double [optimize_norm_euclidian](#) (unsigned int simulation)
- double [optimize_norm_maximum](#) (unsigned int simulation)
- double [optimize_norm_p](#) (unsigned int simulation)
- double [optimize_norm_taxicab](#) (unsigned int simulation)
- void [optimize_print](#) ()
- void [optimize_save_variables](#) (unsigned int simulation, double error)
- void [optimize_best](#) (unsigned int simulation, double value)
- void [optimize_sequential](#) ()
- void * [optimize_thread](#) ([ParallelData](#) *data)
- void [optimize_merge](#) (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)
- void [optimize_synchronise](#) ()
- void [optimize_sweep](#) ()
- void [optimize_MonteCarlo](#) ()
- void [optimize_orthogonal](#) ()
- void [optimize_best_climbing](#) (unsigned int simulation, double value)
- void [optimize_climbing_sequential](#) (unsigned int simulation)
- void * [optimize_climbing_thread](#) ([ParallelData](#) *data)
- double [optimize_estimate_climbing_random](#) (unsigned int variable, unsigned int estimate)
- double [optimize_estimate_climbing_coordinates](#) (unsigned int variable, unsigned int estimate)
- void [optimize_step_climbing](#) (unsigned int simulation)
- void [optimize_climbing](#) ()
- double [optimize_genetic_objective](#) (**Entity** *entity)
- void [optimize_genetic](#) ()
- void [optimize_save_old](#) ()
- void [optimize_merge_old](#) ()
- void [optimize_refine](#) ()
- void [optimize_step](#) ()
- void [optimize_iterate](#) ()
- 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]
- void(* [optimize_algorithm](#))()
Pointer to the function to perform a optimization algorithm step.
- double(* [optimize_estimate_climbing](#))(unsigned int variable, unsigned int estimate)
Pointer to the function to estimate the climbing.
- double(* [optimize_norm](#))(unsigned int simulation)
Pointer to the error norm function.
- [Optimize](#) [optimize](#) [1]
Optimization data.

4.21.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.21.2 Function Documentation

4.21.2.1 optimize_best()

```
void optimize_best (
    unsigned int simulation,
    double value )
```

Function to save the best simulations.

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line [444](#) of file [optimize.c](#).

```
00446 {
00447     unsigned int i, j;
00448     double e;
00449     #if DEBUG_OPTIMIZE
00450     fprintf (stderr, "optimize_best: start\n");
00451     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00452             optimize->nsaveds, optimize->nbest);
00453     #endif
00454     if (optimize->nsaveds < optimize->nbest
00455         || value < optimize->error_best[optimize->nsaveds - 1])
00456     {
00457         if (optimize->nsaveds < optimize->nbest)
00458             ++optimize->nsaveds;
00459         optimize->error_best[optimize->nsaveds - 1] = value;
00460         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00461         for (i = optimize->nsaveds; --i;)
00462         {
00463             if (optimize->error_best[i] < optimize->
00464                 error_best[i - 1])
00465             {
00466                 j = optimize->simulation_best[i];
00467                 e = optimize->error_best[i];
00467                 optimize->simulation_best[i] = optimize->
00467                     simulation_best[i - 1];
```

```

00468         optimize->error_best[i] = optimize->
error_best[i - 1];
00469         optimize->simulation_best[i - 1] = j;
00470         optimize->error_best[i - 1] = e;
00471     }
00472     else
00473         break;
00474 }
00475 }
00476 #if DEBUG_OPTIMIZE
00477 fprintf (stderr, "optimize_best: end\n");
00478 #endif
00479 }

```

4.21.2.2 optimize_best_climbing()

```

void optimize_best_climbing (
    unsigned int simulation,
    double value )

```

Function to save the best simulation in a hill climbing method.

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line [806](#) of file [optimize.c](#).

```

00808 {
00809 #if DEBUG_OPTIMIZE
00810     fprintf (stderr, "optimize_best_climbing: start\n");
00811     fprintf (stderr,
00812         "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00813         simulation, value, optimize->error_best[0]);
00814 #endif
00815     if (value < optimize->error_best[0])
00816     {
00817         optimize->error_best[0] = value;
00818         optimize->simulation_best[0] = simulation;
00819 #if DEBUG_OPTIMIZE
00820         fprintf (stderr,
00821             "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00822             simulation, value);
00823 #endif
00824     }
00825 #if DEBUG_OPTIMIZE
00826     fprintf (stderr, "optimize_best_climbing: end\n");
00827 #endif
00828 }

```

4.21.2.3 optimize_climbing()

```

void optimize_climbing ( )

```

Function to optimize with a hill climbing method.

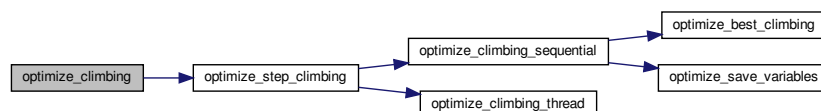
Definition at line [1034](#) of file [optimize.c](#).

```

01035 {
01036     unsigned int i, j, k, b, s, adjust;
01037     #if DEBUG_OPTIMIZE
01038         fprintf (stderr, "optimize_climbing: start\n");
01039     #endif
01040     for (i = 0; i < optimize->nvariables; ++i)
01041         optimize->climbing[i] = 0.;
01042     b = optimize->simulation_best[0] * optimize->
nvariables;
01043     s = optimize->nsimulations;
01044     adjust = 1;
01045     for (i = 0; i < optimize->nsteps; ++i, s += optimize->
nestimates, b = k)
01046     {
01047         #if DEBUG_OPTIMIZE
01048             fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01049                     i, optimize->simulation_best[0]);
01050         #endif
01051         optimize_step_climbing (s);
01052         k = optimize->simulation_best[0] * optimize->
nvariables;
01053         #if DEBUG_OPTIMIZE
01054             fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01055                     i, optimize->simulation_best[0]);
01056         #endif
01057         if (k == b)
01058         {
01059             if (adjust)
01060                 for (j = 0; j < optimize->nvariables; ++j)
01061                     optimize->step[j] *= 0.5;
01062             for (j = 0; j < optimize->nvariables; ++j)
01063                 optimize->climbing[j] = 0.;
01064             adjust = 1;
01065         }
01066         else
01067         {
01068             for (j = 0; j < optimize->nvariables; ++j)
01069                 {
01070                     #if DEBUG_OPTIMIZE
01071                         fprintf (stderr,
01072                                 "optimize_climbing: best%u=%.14le old%u=%.14le\n",
01073                                 j, optimize->value[k + j], j, optimize->
value[b + j]);
01074                     #endif
01075                     optimize->climbing[j]
01076                         = (1. - optimize->relaxation) * optimize->
climbing[j]
01077                         + optimize->relaxation
01078                         * (optimize->value[k + j] - optimize->value[b + j]);
01079                     #if DEBUG_OPTIMIZE
01080                         fprintf (stderr, "optimize_climbing: climbing%u=%.14le\n",
01081                                 j, optimize->climbing[j]);
01082                     #endif
01083                 }
01084             adjust = 0;
01085         }
01086     }
01087     #if DEBUG_OPTIMIZE
01088         fprintf (stderr, "optimize_climbing: end\n");
01089     #endif
01090 }

```

Here is the call graph for this function:



4.21.2.4 optimize_climbing_sequential()

```
void optimize_climbing_sequential (
    unsigned int simulation )
```

Function to estimate the hill climbing sequentially.

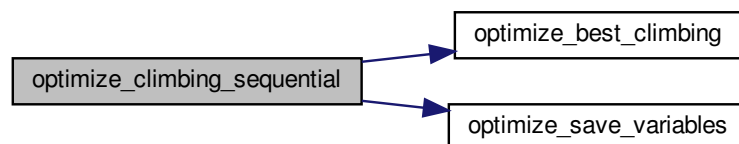
Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 834 of file [optimize.c](#).

```
00835 {
00836     double e;
00837     unsigned int i, j;
00838     #if DEBUG_OPTIMIZE
00839     fprintf (stderr, "optimize_climbing_sequential: start\n");
00840     fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00841             "nend_climbing=%u\n",
00842             optimize->nstart_climbing, optimize->
nend_climbing);
00843     #endif
00844     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00845     {
00846         j = simulation + i;
00847         e = optimize_norm (j);
00848         optimize_best_climbing (j, e);
00849         optimize_save_variables (j, e);
00850         if (e < optimize->threshold)
00851         {
00852             optimize->stop = 1;
00853             break;
00854         }
00855     #if DEBUG_OPTIMIZE
00856     fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
00857     #endif
00858     }
00859     #if DEBUG_OPTIMIZE
00860     fprintf (stderr, "optimize_climbing_sequential: end\n");
00861     #endif
00862 }
```

Here is the call graph for this function:



4.21.2.5 optimize_climbing_thread()

```
void* optimize_climbing_thread (
    ParallelData * data )
```

Function to estimate the hill climbing on a thread.

Returns

NULL

Parameters

<i>data</i>	Function data.
-------------	----------------

Definition at line 870 of file [optimize.c](#).

```
00871 {
00872     unsigned int i, thread;
00873     double e;
00874     #if DEBUG_OPTIMIZE
00875     fprintf (stderr, "optimize_climbing_thread: start\n");
00876     #endif
00877     thread = data->thread;
00878     #if DEBUG_OPTIMIZE
00879     fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00880             thread,
00881             optimize->thread_climbing[thread],
00882             optimize->thread_climbing[thread + 1]);
00883     #endif
00884     for (i = optimize->thread_climbing[thread];
00885          i < optimize->thread_climbing[thread + 1]; ++i)
00886     {
00887         e = optimize_norm (i);
00888         g_mutex_lock (mutex);
00889         optimize_best_climbing (i, e);
00890         optimize_save_variables (i, e);
00891         if (e < optimize->threshold)
00892             optimize->stop = 1;
00893         g_mutex_unlock (mutex);
00894         if (optimize->stop)
00895             break;
00896     #if DEBUG_OPTIMIZE
00897     fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00898     #endif
00899     }
00900     #if DEBUG_OPTIMIZE
00901     fprintf (stderr, "optimize_climbing_thread: end\n");
00902     #endif
00903     g_thread_exit (NULL);
00904     return NULL;
00905 }
```

4.21.2.6 optimize_estimate_climbing_coordinates()

```
double optimize_estimate_climbing_coordinates (
    unsigned int variable,
    unsigned int estimate )
```

Function to estimate a component of the hill climbing vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 935 of file [optimize.c](#).

```

00939 {
00940     double x;
00941     #if DEBUG_OPTIMIZE
00942     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00943     #endif
00944     x = optimize->climbing[variable];
00945     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00946     {
00947         if (estimate & 1)
00948             x += optimize->step[variable];
00949         else
00950             x -= optimize->step[variable];
00951     }
00952     #if DEBUG_OPTIMIZE
00953     fprintf (stderr,
00954             "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
00955             variable, x);
00956     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00957     #endif
00958     return x;
00959 }

```

4.21.2.7 optimize_estimate_climbing_random()

```

double optimize_estimate_climbing_random (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the hill climbing vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 911 of file [optimize.c](#).

```

00916 {
00917     double x;
00918     #if DEBUG_OPTIMIZE
00919     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00920     #endif
00921     x = optimize->climbing[variable]
00922         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00923         step[variable];
00924     #if DEBUG_OPTIMIZE
00925     fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
00926             variable, x);
00927     fprintf (stderr, "optimize_estimate_climbing_random: end\n");
00928     #endif
00929     return x;
00930 }

```

4.21.2.8 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the [Optimize](#) struct.

Definition at line 1391 of file [optimize.c](#).

```
01392 {
01393     unsigned int i, j;
01394     #if DEBUG_OPTIMIZE
01395     fprintf (stderr, "optimize_free: start\n");
01396     #endif
01397     for (j = 0; j < optimize->ninputs; ++j)
01398     {
01399         for (i = 0; i < optimize->nexperiments; ++i)
01400             g_mapped_file_unref (optimize->file[j][i]);
01401         g_free (optimize->file[j]);
01402     }
01403     g_free (optimize->error_old);
01404     g_free (optimize->value_old);
01405     g_free (optimize->value);
01406     g_free (optimize->genetic_variable);
01407     #if DEBUG_OPTIMIZE
01408     fprintf (stderr, "optimize_free: end\n");
01409     #endif
01410 }
```

4.21.2.9 optimize_genetic()

```
void optimize_genetic ( )
```

Function to optimize with the genetic algorithm.

Definition at line 1131 of file [optimize.c](#).

```
01132 {
01133     double *best_variable = NULL;
01134     char *best_genome = NULL;
01135     double best_objective = 0.;
01136     #if DEBUG_OPTIMIZE
01137     fprintf (stderr, "optimize_genetic: start\n");
01138     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01139             nthreads);
01140     fprintf (stderr,
01141             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01142             optimize->nvariables, optimize->
01143             nsimulations, optimize->niterations);
01144     fprintf (stderr,
01145             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01146             optimize->mutation_ratio, optimize->
01147             reproduction_ratio,
01148             optimize->adaptation_ratio);
01149     #endif
01150     genetic_algorithm_default (optimize->nvariables,
01151                               optimize->genetic_variable,
01152                               optimize->nsimulations,
01153                               optimize->niterations,
01154                               optimize->mutation_ratio,
01155                               optimize->reproduction_ratio,
01156                               optimize->adaptation_ratio,
01157                               optimize->seed,
01158                               optimize->threshold,
01159                               &optimize_genetic_objective,
01160                               &best_genome, &best_variable, &best_objective);
01161     #if DEBUG_OPTIMIZE
01162     fprintf (stderr, "optimize_genetic: the best\n");
01163     #endif
01164     optimize->error_old = (double *) g_malloc (sizeof (double));
01165     optimize->value_old
```

```

01164     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01165     optimize->error_old[0] = best_objective;
01166     memcpy (optimize->value_old, best_variable,
01167             optimize->nvariables * sizeof (double));
01168     g_free (best_genome);
01169     g_free (best_variable);
01170     optimize_print ();
01171     #if DEBUG_OPTIMIZE
01172     fprintf (stderr, "optimize_genetic: end\n");
01173     #endif
01174 }

```

4.21.2.10 optimize_genetic_objective()

```

double optimize_genetic_objective (
    Entity * entity )

```

Function to calculate the objective function of an entity.

Returns

objective function value.

Parameters

<i>entity</i>	entity data.
---------------	--------------

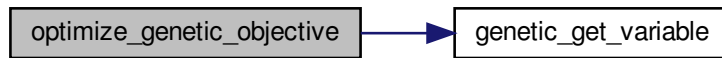
Definition at line 1098 of file [optimize.c](#).

```

01099 {
01100     unsigned int j;
01101     double objective;
01102     char buffer[64];
01103     #if DEBUG_OPTIMIZE
01104     fprintf (stderr, "optimize_genetic_objective: start\n");
01105     #endif
01106     for (j = 0; j < optimize->nvariables; ++j)
01107     {
01108         optimize->value[entity->id * optimize->nvariables + j]
01109         = genetic_get_variable (entity, optimize->genetic_variable + j);
01110     }
01111     objective = optimize_norm (entity->id);
01112     g_mutex_lock (mutex);
01113     for (j = 0; j < optimize->nvariables; ++j)
01114     {
01115         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01116         fprintf (optimize->file_variables, buffer,
01117                 genetic_get_variable (entity, optimize->genetic_variable + j));
01118     }
01119     fprintf (optimize->file_variables, "%.14le\n", objective);
01120     g_mutex_unlock (mutex);
01121     #if DEBUG_OPTIMIZE
01122     fprintf (stderr, "optimize_genetic_objective: end\n");
01123     #endif
01124     return objective;
01125 }

```


Here is the call graph for this function:



4.21.2.11 optimize_input()

```
void optimize_input (
    unsigned int simulation,
    char * input,
    GMappedFile * stencil )
```

Function to write the simulation input file.

Parameters

<i>simulation</i>	Simulation number.
<i>input</i>	Input file name.
<i>stencil</i>	Template of the input file name.

Definition at line 93 of file [optimize.c](#).

```
00096 {
00097     char buffer[32], value[32];
00098     GRegex *regex;
00099     FILE *file;
00100     char *buffer2, *buffer3 = NULL, *content;
00101     gsize length;
00102     unsigned int i;
00103
00104     #if DEBUG_OPTIMIZE
00105     fprintf (stderr, "optimize_input: start\n");
00106     #endif
00107
00108     // Checking the file
00109     if (!stencil)
00110         goto optimize_input_end;
00111
00112     // Opening stencil
00113     content = g_mapped_file_get_contents (stencil);
00114     length = g_mapped_file_get_length (stencil);
00115     #if DEBUG_OPTIMIZE
00116     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00117     #endif
00118     file = g_fopen (input, "w");
00119
00120     // Parsing stencil
00121     for (i = 0; i < optimize->nvariables; ++i)
00122     {
00123         #if DEBUG_OPTIMIZE
00124         fprintf (stderr, "optimize_input: variable=%u\n", i);
00125         #endif
00126         snprintf (buffer, 32, "@variable%u@", i + 1);
00127         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
```

```

00128                                     NULL);
00129         if (i == 0)
00130         {
00131             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00132                                               optimize->label[i],
00133                                               (GRegexMatchFlags) 0, NULL);
00134 #if DEBUG_OPTIMIZE
00135             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00136 #endif
00137         }
00138         else
00139         {
00140             length = strlen (buffer3);
00141             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00142                                               optimize->label[i],
00143                                               (GRegexMatchFlags) 0, NULL);
00144             g_free (buffer3);
00145         }
00146         g_regex_unref (regex);
00147         length = strlen (buffer2);
00148         snprintf (buffer, 32, "@value%u@", i + 1);
00149         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00150                             NULL);
00151         snprintf (value, 32, format[optimize->precision[i]],
00152                  optimize->value[simulation * optimize->
00153                                nvariables + i]);
00154 #if DEBUG_OPTIMIZE
00155         fprintf (stderr, "optimize_input: value=%s\n", value);
00156 #endif
00157         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00158                                           (GRegexMatchFlags) 0, NULL);
00159         g_free (buffer2);
00160         g_regex_unref (regex);
00161     }
00162
00163     // Saving input file
00164     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00165     g_free (buffer3);
00166     fclose (file);
00167
00168 optimize_input_end:
00169 #if DEBUG_OPTIMIZE
00170     fprintf (stderr, "optimize_input: end\n");
00171 #endif
00172     return;
00173 }

```

4.21.2.12 optimize_iterate()

```
void optimize_iterate ( )
```

Function to iterate the algorithm.

Definition at line 1361 of file [optimize.c](#).

```

01362 {
01363     unsigned int i;
01364 #if DEBUG_OPTIMIZE
01365     fprintf (stderr, "optimize_iterate: start\n");
01366 #endif
01367     optimize->error_old = (double *) g_malloc (optimize->
01368         nbest * sizeof (double));
01369     optimize->value_old =
01370         (double *) g_malloc (optimize->nbest * optimize->
01371             nvariables *
01372             sizeof (double));
01373     optimize_step ();
01374     optimize_save_old ();
01375     optimize_refine ();
01376     optimize_print ();
01377     for (i = 1; i < optimize->niterations && !optimize->
01378         stop; ++i)
01379     {
01380         optimize_step ();

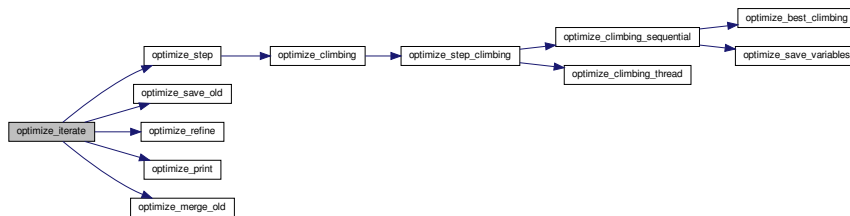
```

```

01378     optimize_merge_old ();
01379     optimize_refine ();
01380     optimize_print ();
01381 }
01382 #if DEBUG_OPTIMIZE
01383     fprintf (stderr, "optimize_iterate: end\n");
01384 #endif
01385 }

```

Here is the call graph for this function:



4.21.2.13 optimize_merge()

```

void optimize_merge (
    unsigned int nsaveds,
    unsigned int * simulation_best,
    double * error_best )

```

Function to merge the 2 optimization results.

Parameters

<i>nsaveds</i>	Number of saved results.
<i>simulation_best</i>	Array of best simulation numbers.
<i>error_best</i>	Array of best objective function values.

Definition at line 557 of file [optimize.c](#).

```

00562 {
00563     unsigned int i, j, k, s[optimize->nbest];
00564     double e[optimize->nbest];
00565     #if DEBUG_OPTIMIZE
00566         fprintf (stderr, "optimize_merge: start\n");
00567     #endif
00568     i = j = k = 0;
00569     do
00570     {
00571         if (i == optimize->nsaveds)
00572         {
00573             s[k] = simulation_best[j];
00574             e[k] = error_best[j];
00575             ++j;
00576             ++k;
00577             if (j == nsaveds)
00578                 break;
00579         }
00580         else if (j == nsaveds)
00581         {

```

```

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

```

4.21.2.14 optimize_merge_old()

```
void optimize_merge_old ( )
```

Function to merge the best results with the previous step best results on iterative methods.

Definition at line 1212 of file [optimize.c](#).

```

01213 {
01214     unsigned int i, j, k;
01215     double v[optimize->nbest * optimize->nvariables], e[
01216         optimize->nbest],
01217         *enew, *eold;
01218     #if DEBUG_OPTIMIZE
01219     fprintf (stderr, "optimize_merge_old: start\n");
01220     #endif
01221     enew = optimize->error_best;
01222     eold = optimize->error_old;
01223     i = j = k = 0;
01224     do
01225     {
01226         if (*enew < *eold)
01227         {
01228             memcpy (v + k * optimize->nvariables,
01229                 optimize->value
01230                 + optimize->simulation_best[i] *
01231                 optimize->nvariables,
01232                 optimize->nvariables * sizeof (double));
01233             e[k] = *enew;
01234             ++k;
01235             ++enew;
01236             ++i;
01237         }
01238         else
01239         {
01240             memcpy (v + k * optimize->nvariables,
01241                 optimize->value_old + j * optimize->
01242                 nvariables,
01243                 optimize->nvariables * sizeof (double));
01244             e[k] = *eold;
01245             ++k;
01246             ++eold;
01247             ++j;
01248         }
01249     }
01250 }

```

```

01246     }
01247     while (k < optimize->nbest);
01248     memcpy (optimize->value_old, v, k * optimize->
nvariables * sizeof (double));
01249     memcpy (optimize->error_old, e, k * sizeof (double));
01250     #if DEBUG_OPTIMIZE
01251     fprintf (stderr, "optimize_merge_old: end\n");
01252     #endif
01253 }

```

4.21.2.15 optimize_MonteCarlo()

```
void optimize_MonteCarlo ( )
```

Function to optimize with the Monte-Carlo algorithm.

Definition at line 715 of file [optimize.c](#).

```

00716 {
00717     unsigned int i, j;
00718     GThread *thread[nthreads];
00719     ParallelData data[nthreads];
00720     #if DEBUG_OPTIMIZE
00721     fprintf (stderr, "optimize_MonteCarlo: start\n");
00722     #endif
00723     for (i = 0; i < optimize->nsimulations; ++i)
00724         for (j = 0; j < optimize->nvariables; ++j)
00725             optimize->value[i * optimize->nvariables + j]
00726                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->
rng)
00727                     * (optimize->rangemax[j] - optimize->rangemin[j]);
00728     optimize->nsaveds = 0;
00729     if (nthreads <= 1)
00730         optimize_sequential ();
00731     else
00732     {
00733         for (i = 0; i < nthreads; ++i)
00734         {
00735             data[i].thread = i;
00736             thread[i]
00737                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00738         }
00739         for (i = 0; i < nthreads; ++i)
00740             g_thread_join (thread[i]);
00741     }
00742     #if HAVE_MPI
00743     // Communicating tasks results
00744     optimize_synchronise ();
00745     #endif
00746     #if DEBUG_OPTIMIZE
00747     fprintf (stderr, "optimize_MonteCarlo: end\n");
00748     #endif
00749 }

```

4.21.2.16 optimize_norm_euclidian()

```
double optimize_norm_euclidian (
    unsigned int simulation )
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

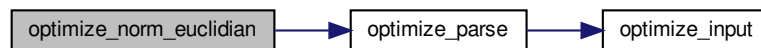
Definition at line 292 of file [optimize.c](#).

```

00293 {
00294     double e, ei;
00295     unsigned int i;
00296     #if DEBUG_OPTIMIZE
00297     fprintf (stderr, "optimize_norm_euclidian: start\n");
00298     #endif
00299     e = 0.;
00300     for (i = 0; i < optimize->nexperiments; ++i)
00301     {
00302         ei = optimize_parse (simulation, i);
00303         e += ei * ei;
00304     }
00305     e = sqrt (e);
00306     #if DEBUG_OPTIMIZE
00307     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00308     fprintf (stderr, "optimize_norm_euclidian: end\n");
00309     #endif
00310     return e;
00311 }

```

Here is the call graph for this function:

**4.21.2.17 optimize_norm_maximum()**

```

double optimize_norm_maximum (
    unsigned int simulation )

```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 319 of file [optimize.c](#).

```

00320 {

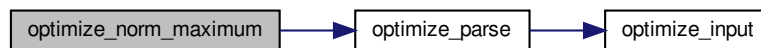
```

```

00321 double e, ei;
00322 unsigned int i;
00323 #if DEBUG_OPTIMIZE
00324 fprintf (stderr, "optimize_norm_maximum: start\n");
00325 #endif
00326 e = 0.;
00327 for (i = 0; i < optimize->nexperiments; ++i)
00328 {
00329     ei = fabs (optimize_parse (simulation, i));
00330     e = fmax (e, ei);
00331 }
00332 #if DEBUG_OPTIMIZE
00333 fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00334 fprintf (stderr, "optimize_norm_maximum: end\n");
00335 #endif
00336 return e;
00337 }

```

Here is the call graph for this function:



4.21.2.18 optimize_norm_p()

```

double optimize_norm_p (
    unsigned int simulation )

```

Function to calculate the P error norm.

Returns

P error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 345 of file [optimize.c](#).

```

00346 {
00347     double e, ei;
00348     unsigned int i;
00349     #if DEBUG_OPTIMIZE
00350     fprintf (stderr, "optimize_norm_p: start\n");
00351     #endif
00352     e = 0.;
00353     for (i = 0; i < optimize->nexperiments; ++i)
00354     {
00355         ei = fabs (optimize_parse (simulation, i));
00356         e += pow (ei, optimize->p);
00357     }
00358     e = pow (e, 1. / optimize->p);
00359     #if DEBUG_OPTIMIZE

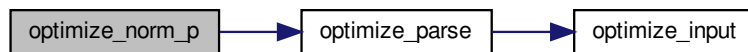
```

```

00360     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00361     fprintf (stderr, "optimize_norm_p: end\n");
00362 #endif
00363     return e;
00364 }

```

Here is the call graph for this function:



4.21.2.19 optimize_norm_taxicab()

```

double optimize_norm_taxicab (
    unsigned int simulation )

```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

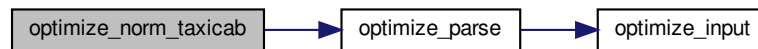
Definition at line 372 of file [optimize.c](#).

```

00373 {
00374     double e;
00375     unsigned int i;
00376     #if DEBUG_OPTIMIZE
00377     fprintf (stderr, "optimize_norm_taxicab: start\n");
00378     #endif
00379     e = 0.;
00380     for (i = 0; i < optimize->nexperiments; ++i)
00381         e += fabs (optimize_parse (simulation, i));
00382     #if DEBUG_OPTIMIZE
00383     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00384     fprintf (stderr, "optimize_norm_taxicab: end\n");
00385     #endif
00386     return e;
00387 }

```


Here is the call graph for this function:



4.21.2.20 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1416 of file [optimize.c](#).

```

01417 {
01418     GTimeZone *tz;
01419     GDateTime *t0, *t;
01420     unsigned int i, j;
01421
01422     #if DEBUG_OPTIMIZE
01423         char *buffer;
01424         fprintf (stderr, "optimize_open: start\n");
01425     #endif
01426
01427     // Getting initial time
01428     #if DEBUG_OPTIMIZE
01429         fprintf (stderr, "optimize_open: getting initial time\n");
01430     #endif
01431     tz = g_time_zone_new_utc ();
01432     t0 = g_date_time_new_now (tz);
01433
01434     // Obtaining and initing the pseudo-random numbers generator seed
01435     #if DEBUG_OPTIMIZE
01436         fprintf (stderr, "optimize_open: getting initial seed\n");
01437     #endif
01438     if (optimize->seed == DEFAULT_RANDOM_SEED)
01439         optimize->seed = input->seed;
01440     gsl_rng_set (optimize->rng, optimize->seed);
01441
01442     // Replacing the working directory
01443     #if DEBUG_OPTIMIZE
01444         fprintf (stderr, "optimize_open: replacing the working directory\n");
01445     #endif
01446     g_chdir (input->directory);
01447
01448     // Getting results file names
01449     optimize->result = input->result;
01450     optimize->variables = input->variables;
01451
01452     // Obtaining the simulator file
01453     optimize->simulator = input->simulator;
01454
01455     // Obtaining the evaluator file
01456     optimize->evaluator = input->evaluator;
01457
01458     // Reading the algorithm
01459     optimize->algorithm = input->algorithm;
01460     switch (optimize->algorithm)
01461     {
01462         case ALGORITHM_MONTE_CARLO:
01463             optimize_algorithm = optimize_MonteCarlo;
01464             break;
01465         case ALGORITHM_SWEEP:
01466             optimize_algorithm = optimize_sweep;
01467             break;
    
```

```

01468     case ALGORITHM_ORTHOGONAL:
01469         optimize_algorithm = optimize_orthogonal;
01470         break;
01471     default:
01472         optimize_algorithm = optimize_genetic;
01473         optimize->mutation_ratio = input->
mutation_ratio;
01474         optimize->reproduction_ratio = input->
reproduction_ratio;
01475         optimize->adaptation_ratio = input->
adaptation_ratio;
01476     }
01477     optimize->nvariables = input->nvariables;
01478     optimize->nsimulations = input->nsimulations;
01479     optimize->niterations = input->niterations;
01480     optimize->nbest = input->nbest;
01481     optimize->tolerance = input->tolerance;
01482     optimize->nsteps = input->nsteps;
01483     optimize->nestimates = 0;
01484     optimize->threshold = input->threshold;
01485     optimize->stop = 0;
01486     if (input->nsteps)
01487     {
01488         optimize->relaxation = input->relaxation;
01489         switch (input->climbing)
01490         {
01491             case CLIMBING_METHOD_COORDINATES:
01492                 optimize->nestimates = 2 * optimize->
nvariables;
01493                 optimize_estimate_climbing =
optimize_estimate_climbing_coordinates;
01494                 break;
01495             default:
01496                 optimize->nestimates = input->nestimates;
01497                 optimize_estimate_climbing =
optimize_estimate_climbing_random;
01498         }
01499     }
01500
01501 #if DEBUG_OPTIMIZE
01502 fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01503 #endif
01504 optimize->simulation_best
01505     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01506 optimize->error_best = (double *) alloca (optimize->
nbest * sizeof (double));
01507
01508 // Reading the experimental data
01509 #if DEBUG_OPTIMIZE
01510 buffer = g_get_current_dir ();
01511 fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01512 g_free (buffer);
01513 #endif
01514 optimize->nexperiments = input->nexperiments;
01515 optimize->ninputs = input->experiment->ninputs;
01516 optimize->experiment
01517     = (char **) alloca (input->nexperiments * sizeof (char *));
01518 optimize->weight = (double *) alloca (input->nexperiments * sizeof (double
));
01519 for (i = 0; i < input->experiment->ninputs; ++i)
01520     optimize->file[i] = (GMappedFile **)
01521         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01522 for (i = 0; i < input->nexperiments; ++i)
01523     {
01524 #if DEBUG_OPTIMIZE
01525         fprintf (stderr, "optimize_open: i=%u\n", i);
01526 #endif
01527         optimize->experiment[i] = input->experiment[i].
name;
01528         optimize->weight[i] = input->experiment[i].
weight;
01529 #if DEBUG_OPTIMIZE
01530         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
optimize->experiment[i], optimize->
weight[i]);
01531 #endif
01532         for (j = 0; j < input->experiment->ninputs; ++j)
01533             {
01534 #if DEBUG_OPTIMIZE
01535                 fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01536 #endif
01537                 optimize->file[j][i]
01538                     = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01539             }
01540     }
01541
01542
01543 // Reading the variables data

```

```

01544 #if DEBUG_OPTIMIZE
01545     fprintf (stderr, "optimize_open: reading variables\n");
01546 #endif
01547     optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01548     j = input->nvariables * sizeof (double);
01549     optimize->rangemin = (double *) alloca (j);
01550     optimize->rangeminabs = (double *) alloca (j);
01551     optimize->rangemax = (double *) alloca (j);
01552     optimize->rangemaxabs = (double *) alloca (j);
01553     optimize->step = (double *) alloca (j);
01554     j = input->nvariables * sizeof (unsigned int);
01555     optimize->precision = (unsigned int *) alloca (j);
01556     optimize->nsweeps = (unsigned int *) alloca (j);
01557     optimize->nbits = (unsigned int *) alloca (j);
01558     for (i = 0; i < input->nvariables; ++i)
01559     {
01560         optimize->label[i] = input->variable[i].name;
01561         optimize->rangemin[i] = input->variable[i].
rangemin;
01562         optimize->rangeminabs[i] = input->variable[i].
rangeminabs;
01563         optimize->rangemax[i] = input->variable[i].
rangemax;
01564         optimize->rangemaxabs[i] = input->variable[i].
rangemaxabs;
01565         optimize->precision[i] = input->variable[i].
precision;
01566         optimize->step[i] = input->variable[i].step;
01567         optimize->nsweeps[i] = input->variable[i].
nsweeps;
01568         optimize->nbits[i] = input->variable[i].nbits;
01569     }
01570     if (input->algorithm == ALGORITHM_SWEEP
01571         || input->algorithm == ALGORITHM_ORTHOGONAL)
01572     {
01573         optimize->nsimulations = 1;
01574         for (i = 0; i < input->nvariables; ++i)
01575         {
01576             optimize->nsimulations *= optimize->
nsweeps[i];
01577 #if DEBUG_OPTIMIZE
01578             fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01579                     optimize->nsweeps[i], optimize->
nsimulations);
01580 #endif
01581         }
01582     }
01583     if (optimize->nsteps)
01584         optimize->climbing
01585         = (double *) alloca (optimize->nvariables * sizeof (double));
01586
01587     // Setting error norm
01588     switch (input->norm)
01589     {
01590     case ERROR_NORM_EUCLIDIAN:
01591         optimize_norm = optimize_norm_euclidian;
01592         break;
01593     case ERROR_NORM_MAXIMUM:
01594         optimize_norm = optimize_norm_maximum;
01595         break;
01596     case ERROR_NORM_P:
01597         optimize_norm = optimize_norm_p;
01598         optimize->p = input->p;
01599         break;
01600     default:
01601         optimize_norm = optimize_norm_taxicab;
01602     }
01603
01604     // Allocating values
01605 #if DEBUG_OPTIMIZE
01606     fprintf (stderr, "optimize_open: allocating variables\n");
01607     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01608             optimize->nvariables, optimize->algorithm);
01609 #endif
01610     optimize->genetic_variable = NULL;
01611     if (optimize->algorithm == ALGORITHM_GENETIC)
01612     {
01613         optimize->genetic_variable = (GeneticVariable *)
01614         g_malloc (optimize->nvariables * sizeof (
GeneticVariable));
01615         for (i = 0; i < optimize->nvariables; ++i)
01616         {
01617 #if DEBUG_OPTIMIZE
01618             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01619                     i, optimize->rangemin[i], optimize->
rangemax[i],
01620                     optimize->nbits[i]);

```

```

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

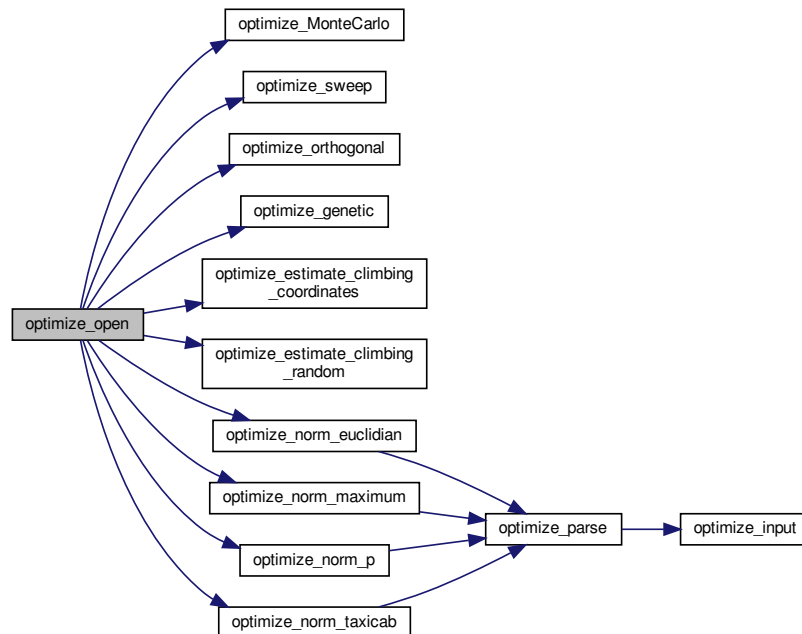
```

```

01696     }
01697
01698     // Getting calculation time
01699     t = g_date_time_new_now (tz);
01700     optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01701     g_date_time_unref (t);
01702     g_date_time_unref (t0);
01703     g_time_zone_unref (tz);
01704     printf ("%s = %.6lg s\n", _("Calculation time"), optimize->
calculation_time);
01705     fprintf (optimize->file_result, "%s = %.6lg s\n",
01706             _("Calculation time"), optimize->calculation_time);
01707
01708     // Closing result files
01709     fclose (optimize->file_variables);
01710     fclose (optimize->file_result);
01711
01712 #if DEBUG_OPTIMIZE
01713     fprintf (stderr, "optimize_open: end\n");
01714 #endif
01715 }

```

Here is the call graph for this function:



4.21.2.21 optimize_orthogonal()

```
void optimize_orthogonal ( )
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 755 of file [optimize.c](#).

```

00756 {
00757     unsigned int i, j, k, l;
00758     double e;
00759     GThread *thread[nthreads];
00760     ParallelData data[nthreads];
00761     #if DEBUG_OPTIMIZE
00762     fprintf (stderr, "optimize_orthogonal: start\n");
00763     #endif
00764     for (i = 0; i < optimize->nsimulations; ++i)
00765     {
00766         k = i;
00767         for (j = 0; j < optimize->nvariables; ++j)
00768         {
00769             l = k % optimize->nsweeps[j];
00770             k /= optimize->nsweeps[j];
00771             e = optimize->rangemin[j];
00772             if (optimize->nsweeps[j] > 1)
00773                 e += (l + gsl_rng_uniform (optimize->rng))
00774                     * (optimize->rangemax[j] - optimize->
rangemin[j]);
00775             / optimize->nsweeps[j];
00776             optimize->value[i * optimize->nvariables + j] = e;
00777         }
00778     }
00779     optimize->nsaveds = 0;
00780     if (nthreads <= 1)
00781         optimize_sequential ();
00782     else
00783     {
00784         for (i = 0; i < nthreads; ++i)
00785         {
00786             data[i].thread = i;
00787             thread[i]
                = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00789         }
00790         for (i = 0; i < nthreads; ++i)
00791             g_thread_join (thread[i]);
00792     }
00793     #if HAVE_MPI
00794     // Communicating tasks results
00795     optimize_synchronize ();
00796     #endif
00797     #if DEBUG_OPTIMIZE
00798     fprintf (stderr, "optimize_orthogonal: end\n");
00799     #endif
00800 }

```

4.21.2.22 optimize_parse()

```

double optimize_parse (
    unsigned int simulation,
    unsigned int experiment )

```

Function to parse input files, simulating and calculating the objective function.

Returns

Objective function value.

Parameters

<i>simulation</i>	Simulation number.
<i>experiment</i>	Experiment number.

Definition at line 182 of file `optimize.c`.

```

00184 {

```

```

00185     unsigned int i;
00186     double e;
00187     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00188         *buffer3, *buffer4;
00189     FILE *file_result;
00190
00191     #if DEBUG_OPTIMIZE
00192         fprintf (stderr, "optimize_parse: start\n");
00193         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00194             simulation, experiment);
00195     #endif
00196
00197     // Opening input files
00198     for (i = 0; i < optimize->ninputs; ++i)
00199     {
00200         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00201         #if DEBUG_OPTIMIZE
00202             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00203         #endif
00204         optimize_input (simulation, &input[i][0], optimize->
00205             file[i][experiment]);
00206         for (; i < MAX_NINPUTS; ++i)
00207             strcpy (&input[i][0], "");
00208         #if DEBUG_OPTIMIZE
00209             fprintf (stderr, "optimize_parse: parsing end\n");
00210         #endif
00211
00212         // Performing the simulation
00213         snprintf (output, 32, "output-%u-%u", simulation, experiment);
00214         buffer2 = g_path_get_dirname (optimize->simulator);
00215         buffer3 = g_path_get_basename (optimize->simulator);
00216         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00217         snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s",
00218             buffer4, input[0], input[1], input[2], input[3], input[4],
00219             input[5], input[6], input[7], output);
00220         g_free (buffer4);
00221         g_free (buffer3);
00222         g_free (buffer2);
00223         #if DEBUG_OPTIMIZE
00224             fprintf (stderr, "optimize_parse: %s\n", buffer);
00225         #endif
00226         system (buffer);
00227
00228         // Checking the objective value function
00229         if (optimize->evaluator)
00230         {
00231             snprintf (result, 32, "result-%u-%u", simulation, experiment);
00232             buffer2 = g_path_get_dirname (optimize->evaluator);
00233             buffer3 = g_path_get_basename (optimize->evaluator);
00234             buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235             snprintf (buffer, 512, "\"%s\" %s %s %s",
00236                 buffer4, output, optimize->experiment[experiment], result);
00237             g_free (buffer4);
00238             g_free (buffer3);
00239             g_free (buffer2);
00240             #if DEBUG_OPTIMIZE
00241                 fprintf (stderr, "optimize_parse: %s\n", buffer);
00242                 fprintf (stderr, "optimize_parse: result=%s\n", result);
00243             #endif
00244             system (buffer);
00245             file_result = g_fopen (result, "r");
00246             e = atof (fgets (buffer, 512, file_result));
00247             fclose (file_result);
00248         }
00249         else
00250         {
00251             #if DEBUG_OPTIMIZE
00252                 fprintf (stderr, "optimize_parse: output=%s\n", output);
00253             #endif
00254             strcpy (result, "");
00255             file_result = g_fopen (output, "r");
00256             e = atof (fgets (buffer, 512, file_result));
00257             fclose (file_result);
00258         }
00259
00260         // Removing files
00261         #if !DEBUG_OPTIMIZE
00262             for (i = 0; i < optimize->ninputs; ++i)
00263             {
00264                 if (optimize->file[i][0])
00265                 {
00266                     snprintf (buffer, 512, RM " %s", &input[i][0]);
00267                     system (buffer);
00268                 }
00269             }
00270             snprintf (buffer, 512, RM " %s %s", output, result);

```

```

00271     system (buffer);
00272 #endif
00273
00274     // Processing pending events
00275     if (show_pending)
00276         show_pending ();
00277
00278 #if DEBUG_OPTIMIZE
00279     fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282     // Returning the objective function
00283     return e * optimize->weight[experiment];
00284 }

```

Here is the call graph for this function:



4.21.2.23 optimize_print()

```
void optimize_print ( )
```

Function to print the results.

Definition at line 393 of file [optimize.c](#).

```

00394 {
00395     unsigned int i;
00396     char buffer[512];
00397     #if HAVE_MPI
00398     if (optimize->mpi_rank)
00399         return;
00400 #endif
00401     printf ("%s\n", _("Best result"));
00402     fprintf (optimize->file_result, "%s\n", _("Best result"));
00403     printf ("error = %.15le\n", optimize->error_old[0]);
00404     fprintf (optimize->file_result, "error = %.15le\n",
00405             optimize->error_old[0]);
00406     for (i = 0; i < optimize->nvariables; ++i)
00407     {
00408         snprintf (buffer, 512, "%s = %s\n",
00409                 optimize->label[i], format[optimize->
00410 precision[i]]);
00409         printf (buffer, optimize->value_old[i]);
00410         fprintf (optimize->file_result, buffer, optimize->
00411 value_old[i]);
00411     }
00412     fflush (optimize->file_result);
00413 }

```


4.21.2.24 optimize_refine()

```
void optimize_refine ( )
```

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

Definition at line 1260 of file [optimize.c](#).

```
01261 {
01262     unsigned int i, j;
01263     double d;
01264     #if HAVE_MPI
01265     MPI_Status mpi_stat;
01266     #endif
01267     #if DEBUG_OPTIMIZE
01268     fprintf (stderr, "optimize_refine: start\n");
01269     #endif
01270     #if HAVE_MPI
01271     if (!optimize->mpi_rank)
01272     {
01273     #endif
01274         for (j = 0; j < optimize->nvariables; ++j)
01275         {
01276             optimize->rangemin[j] = optimize->rangemax[j]
01277             = optimize->value_old[j];
01278         }
01279         for (i = 0; ++i < optimize->nbest;)
01280         {
01281             for (j = 0; j < optimize->nvariables; ++j)
01282             {
01283                 optimize->rangemin[j]
01284                 = fmin (optimize->rangemin[j],
01285                     optimize->value_old[i * optimize->
01286                     nvariables + j]);
01287                 optimize->rangemax[j]
01288                 = fmax (optimize->rangemax[j],
01289                     optimize->value_old[i * optimize->
01290                     nvariables + j]);
01291             }
01292             for (j = 0; j < optimize->nvariables; ++j)
01293             {
01294                 d = optimize->tolerance
01295                 * (optimize->rangemax[j] - optimize->
01296                 rangemin[j]);
01297                 switch (optimize->algorithm)
01298                 {
01299                     case ALGORITHM_MONTE_CARLO:
01300                     {
01301                         d *= 0.5;
01302                         break;
01303                     }
01304                     default:
01305                     {
01306                         if (optimize->nsweeps[j] > 1)
01307                             d /= optimize->nsweeps[j] - 1;
01308                         else
01309                             d = 0.;
01310                     }
01311                 }
01312                 optimize->rangemin[j] -= d;
01313                 optimize->rangemin[j]
01314                 = fmax (optimize->rangemin[j], optimize->
01315                 rangeminabs[j]);
01316                 optimize->rangemax[j] += d;
01317                 optimize->rangemax[j]
01318                 = fmin (optimize->rangemax[j], optimize->
01319                 rangemaxabs[j]);
01320                 printf ("%s min=%lg max=%lg\n", optimize->label[j],
01321                     optimize->rangemin[j], optimize->
01322                     rangemax[j]);
01323                 fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01324                     optimize->label[j], optimize->rangemin[j],
01325                     optimize->rangemax[j]);
01326             }
01327         }
01328     #if HAVE_MPI
01329     for (i = 1; i < ntasks; ++i)
01330     {
01331         MPI_Send (optimize->rangemin, optimize->
01332             nvariables, MPI_DOUBLE, i,
01333             1, MPI_COMM_WORLD);
01334         MPI_Send (optimize->rangemax, optimize->
01335             nvariables, MPI_DOUBLE, i,
01336             1, MPI_COMM_WORLD);
01337     }
01338 }
```

```

01327     else
01328     {
01329         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
01330             1,
01331             MPI_COMM_WORLD, &mpi_stat);
01332         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
01333             1,
01334             MPI_COMM_WORLD, &mpi_stat);
01335     }
01336 #endif
01337 #if DEBUG_OPTIMIZE
01338     fprintf (stderr, "optimize_refine: end\n");
01339 #endif
01340 }

```

4.21.2.25 optimize_save_old()

```
void optimize_save_old ( )
```

Function to save the best results on iterative methods.

Definition at line 1180 of file [optimize.c](#).

```

01181 {
01182     unsigned int i, j;
01183     #if DEBUG_OPTIMIZE
01184         fprintf (stderr, "optimize_save_old: start\n");
01185         fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01186     #endif
01187     memcpy (optimize->error_old, optimize->error_best,
01188         optimize->nbest * sizeof (double));
01189     for (i = 0; i < optimize->nbest; ++i)
01190     {
01191         j = optimize->simulation_best[i];
01192     #if DEBUG_OPTIMIZE
01193         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01194     #endif
01195     }
01196     memcpy (optimize->value_old + i * optimize->
01197         nvariables,
01198         optimize->value + j * optimize->nvariables,
01199         optimize->nvariables * sizeof (double));
01200     }
01201 #if DEBUG_OPTIMIZE
01202     for (i = 0; i < optimize->nvariables; ++i)
01203         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01204             i, optimize->value_old[i]);
01205     fprintf (stderr, "optimize_save_old: end\n");
01206 #endif
01207 }

```

4.21.2.26 optimize_save_variables()

```
void optimize_save_variables (
    unsigned int simulation,
    double error )
```

Function to save in a file the variables and the error.

Parameters

<i>simulation</i>	Simulation number.
<i>error</i>	Error value.

Definition at line 419 of file `optimize.c`.

```

00421 {
00422     unsigned int i;
00423     char buffer[64];
00424     #if DEBUG_OPTIMIZE
00425     fprintf (stderr, "optimize_save_variables: start\n");
00426     #endif
00427     for (i = 0; i < optimize->nvariables; ++i)
00428     {
00429         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00430         fprintf (optimize->file_variables, buffer,
00431                 optimize->value[simulation * optimize->
nvariables + i]);
00432     }
00433     fprintf (optimize->file_variables, "%.14le\n", error);
00434     fflush (optimize->file_variables);
00435     #if DEBUG_OPTIMIZE
00436     fprintf (stderr, "optimize_save_variables: end\n");
00437     #endif
00438 }

```

4.21.2.27 optimize_sequential()

```
void optimize_sequential ( )
```

Function to optimize sequentially.

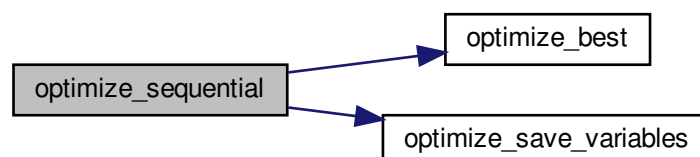
Definition at line 485 of file `optimize.c`.

```

00486 {
00487     unsigned int i;
00488     double e;
00489     #if DEBUG_OPTIMIZE
00490     fprintf (stderr, "optimize_sequential: start\n");
00491     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00492             optimize->nstart, optimize->nend);
00493     #endif
00494     for (i = optimize->nstart; i < optimize->nend; ++i)
00495     {
00496         e = optimize_norm (i);
00497         optimize_best (i, e);
00498         optimize_save_variables (i, e);
00499         if (e < optimize->threshold)
00500         {
00501             optimize->stop = 1;
00502             break;
00503         }
00504     #if DEBUG_OPTIMIZE
00505     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00506     #endif
00507     }
00508     #if DEBUG_OPTIMIZE
00509     fprintf (stderr, "optimize_sequential: end\n");
00510     #endif
00511 }

```

Here is the call graph for this function:



4.21.2.28 optimize_step()

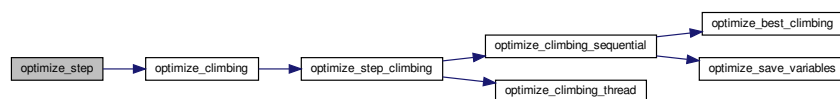
```
void optimize_step ( )
```

Function to do a step of the iterative algorithm.

Definition at line 1344 of file [optimize.c](#).

```
01345 {
01346     #if DEBUG_OPTIMIZE
01347         fprintf (stderr, "optimize_step: start\n");
01348     #endif
01349     optimize_algorithm ();
01350     if (optimize->nsteps)
01351         optimize_climbing ();
01352     #if DEBUG_OPTIMIZE
01353         fprintf (stderr, "optimize_step: end\n");
01354     #endif
01355 }
```

Here is the call graph for this function:



4.21.2.29 optimize_step_climbing()

```
void optimize_step_climbing (
    unsigned int simulation )
```

Function to do a step of the hill climbing method.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 965 of file [optimize.c](#).

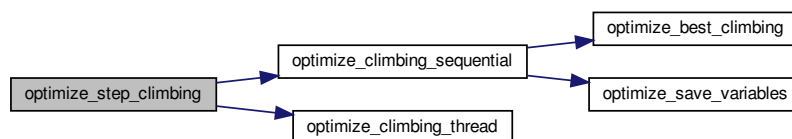
```
00966 {
00967     GThread *thread[nthreads_climbing];
00968     ParallelData data[nthreads_climbing];
00969     unsigned int i, j, k, b;
00970     #if DEBUG_OPTIMIZE
00971         fprintf (stderr, "optimize_step_climbing: start\n");
00972     #endif
00973     for (i = 0; i < optimize->nestimates; ++i)
00974     {
```

```

00975     k = (simulation + i) * optimize->nvariables;
00976     b = optimize->simulation_best[0] * optimize->
nvariables;
00977 #if DEBUG_OPTIMIZE
00978     fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
00979             simulation + i, optimize->simulation_best[0]);
00980 #endif
00981     for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00982     {
00983 #if DEBUG_OPTIMIZE
00984         fprintf (stderr,
00985                 "optimize_step_climbing: estimate=%u best%u=%.14le\n",
00986                 i, j, optimize->value[b]);
00987 #endif
00988         optimize->value[k]
00989             = optimize->value[b] + optimize_estimate_climbing (j, i)
;
00990         optimize->value[k] = fmin (fmax (optimize->value[k],
00991                                         optimize->rangeminabs[j]),
00992                                   optimize->rangemaxabs[j]);
00993 #if DEBUG_OPTIMIZE
00994         fprintf (stderr,
00995                 "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
00996                 i, j, optimize->value[k]);
00997 #endif
00998     }
00999 }
01000 if (nthreads_climbing == 1)
01001     optimize_climbing_sequential (simulation);
01002 else
01003 {
01004     for (i = 0; i <= nthreads_climbing; ++i)
01005     {
01006         optimize->thread_climbing[i]
01007             = simulation + optimize->nstart_climbing
01008             + i * (optimize->nend_climbing - optimize->
nstart_climbing)
01009             / nthreads_climbing;
01010 #if DEBUG_OPTIMIZE
01011         fprintf (stderr,
01012                 "optimize_step_climbing: i=%u thread_climbing=%u\n",
01013                 i, optimize->thread_climbing[i]);
01014 #endif
01015     }
01016     for (i = 0; i < nthreads_climbing; ++i)
01017     {
01018         data[i].thread = i;
01019         thread[i] = g_thread_new
01020             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01021     }
01022     for (i = 0; i < nthreads_climbing; ++i)
01023         g_thread_join (thread[i]);
01024 }
01025 #if DEBUG_OPTIMIZE
01026 fprintf (stderr, "optimize_step_climbing: end\n");
01027 #endif
01028 }

```

Here is the call graph for this function:



4.21.2.30 optimize_sweep()

```
void optimize_sweep ( )
```

Function to optimize with the sweep algorithm.

Definition at line 665 of file [optimize.c](#).

```

00666 {
00667     unsigned int i, j, k, l;
00668     double e;
00669     GThread *thread[nthreads];
00670     ParallelData data[nthreads];
00671     #if DEBUG_OPTIMIZE
00672     fprintf (stderr, "optimize_sweep: start\n");
00673     #endif
00674     for (i = 0; i < optimize->nsimulations; ++i)
00675     {
00676         k = i;
00677         for (j = 0; j < optimize->nvariables; ++j)
00678         {
00679             l = k % optimize->nsweeps[j];
00680             k /= optimize->nsweeps[j];
00681             e = optimize->rangemin[j];
00682             if (optimize->nsweeps[j] > 1)
00683                 e += 1 * (optimize->rangemax[j] - optimize->
rangemin[j])
00684                     / (optimize->nsweeps[j] - 1);
00685             optimize->value[i * optimize->nvariables + j] = e;
00686         }
00687     }
00688     optimize->nsaveds = 0;
00689     if (nthreads <= 1)
00690         optimize_sequential ();
00691     else
00692     {
00693         for (i = 0; i < nthreads; ++i)
00694         {
00695             data[i].thread = i;
00696             thread[i]
= g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00698         }
00699         for (i = 0; i < nthreads; ++i)
00700             g_thread_join (thread[i]);
00701     }
00702     #if HAVE_MPI
00703     // Communicating tasks results
00704     optimize_synchronise ();
00705     #endif
00706     #if DEBUG_OPTIMIZE
00707     fprintf (stderr, "optimize_sweep: end\n");
00708     #endif
00709 }

```

4.21.2.31 optimize_synchronise()

```
void optimize_synchronise ( )
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 618 of file [optimize.c](#).

```

00619 {
00620     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00621     double error_best[optimize->nbest];
00622     MPI_Status mpi_stat;
00623     #if DEBUG_OPTIMIZE
00624     fprintf (stderr, "optimize_synchronise: start\n");
00625     #endif
00626     if (optimize->mpi_rank == 0)
00627     {
00628         for (i = 1; i < ntasks; ++i)
00629         {
00630             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00631             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
MPI_COMM_WORLD, &mpi_stat);
00632

```

```

00633     MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00634              MPI_COMM_WORLD, &mpi_stat);
00635     optimize_merge (nsaveds, simulation_best, error_best);
00636     MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637     if (stop)
00638         optimize->stop = 1;
00639 }
00640 for (i = 1; i < ntasks; ++i)
00641     MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00642 }
00643 else
00644 {
00645     MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00646     MPI_Send (optimize->simulation_best, optimize->
nsaveds, MPI_INT, 0, 1,
00647              MPI_COMM_WORLD);
00648     MPI_Send (optimize->error_best, optimize->
nsaveds, MPI_DOUBLE, 0, 1,
00649              MPI_COMM_WORLD);
00650     MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00651     MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00652     if (stop)
00653         optimize->stop = 1;
00654 }
00655 #if DEBUG_OPTIMIZE
00656 fprintf (stderr, "optimize_synchronise: end\n");
00657 #endif
00658 }

```

4.21.2.32 optimize_thread()

```

void* optimize_thread (
    ParallelData * data )

```

Function to optimize on a thread.

Returns

NULL.

Parameters

<i>data</i>	Function data.
-------------	----------------

Definition at line 519 of file [optimize.c](#).

```

00520 {
00521     unsigned int i, thread;
00522     double e;
00523     #if DEBUG_OPTIMIZE
00524     fprintf (stderr, "optimize_thread: start\n");
00525     #endif
00526     thread = data->thread;
00527     #if DEBUG_OPTIMIZE
00528     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00529             optimize->thread[thread], optimize->thread[thread + 1]);
00530     #endif
00531     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00532     {
00533         e = optimize_norm (i);
00534         g_mutex_lock (mutex);
00535         optimize_best (i, e);
00536         optimize_save_variables (i, e);
00537         if (e < optimize->threshold)
00538             optimize->stop = 1;
00539         g_mutex_unlock (mutex);

```

```

00540         if (optimize->stop)
00541             break;
00542 #if DEBUG_OPTIMIZE
00543     fprintf(stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00544 #endif
00545     }
00546 #if DEBUG_OPTIMIZE
00547     fprintf(stderr, "optimize_thread: end\n");
00548 #endif
00549     g_thread_exit (NULL);
00550     return NULL;
00551 }

```

4.22 optimize.h

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef OPTIMIZE__H
00033 #define OPTIMIZE__H 1
00034
00035 typedef struct
00036 {
00037     GMappedFile **file[MAX_NINPUTS];
00038     char **experiment;
00039     char **label;
00040     gsl_rng *rng;
00041     GeneticVariable *genetic_variable;
00042     FILE *file_result;
00043     FILE *file_variables;
00044     char *result;
00045     char *variables;
00046     char *simulator;
00047     char *evaluator;
00048     double *value;
00049     double *rangemin;
00050     double *rangemax;
00051     double *rangeminabs;
00052     double *rangemaxabs;
00053     double *error_best;
00054     double *weight;
00055     double *step;
00056     double *climbing;
00057     double *value_old;
00058     double *error_old;
00059     unsigned int *precision;
00060     unsigned int *nsweeps;
00061     unsigned int *nbits;
00062     unsigned int *thread;
00063     unsigned int *thread_climbing;
00064     unsigned int *simulation_best;
00065     double tolerance;
00066     double mutation_ratio;
00067     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 void (*optimize_algorithm) ();
00132 extern double (*optimize_estimate_climbing) (unsigned int variable,
00133                                              unsigned int estimate);
00134 extern double (*optimize_norm) (unsigned int simulation);
00135 extern Optimize optimize[1];
00136
00137 // Public functions
00138 void optimize_input (unsigned int simulation, char *input,
00139                     GMappedFile * stencil);
00140 double optimize_parse (unsigned int simulation, unsigned int experiment);
00141 double optimize_norm_euclidian (unsigned int simulation);
00142 double optimize_norm_maximum (unsigned int simulation);
00143 double optimize_norm_p (unsigned int simulation);
00144 double optimize_norm_taxicab (unsigned int simulation);
00145 void optimize_print ();
00146 void optimize_save_variables (unsigned int simulation, double error);
00147 void optimize_best (unsigned int simulation, double value);
00148 void optimize_sequential ();
00149 void *optimize_thread (ParallelData * data);
00150 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00151                    double *error_best);
00152 #if HAVE_MPI
00153 void optimize_synchronise ();
00154 #endif
00155 void optimize_sweep ();
00156 void optimize_MonteCarlo ();
00157 void optimize_orthogonal ();
00158 void optimize_best_climbing (unsigned int simulation, double value);
00159 void optimize_climbing_sequential (unsigned int simulation);
00160 void *optimize_climbing_thread (ParallelData * data);
00161 double optimize_estimate_climbing_random (unsigned int variable,
00162                                           unsigned int estimate);
00163 double optimize_estimate_climbing_coordinates (unsigned int variable,
00164                                              unsigned int estimate);
00165 void optimize_step_climbing (unsigned int simulation);
00166 void optimize_climbing ();
00167 double optimize_genetic_objective (Entity * entity);
00168 void optimize_genetic ();
00169 void optimize_save_old ();
00170 void optimize_merge_old ();
00171 void optimize_refine ();
00172 void optimize_step ();
00173 void optimize_iterate ();
00174 void optimize_free ();
00175 void optimize_open ();
00176
00177 #endif

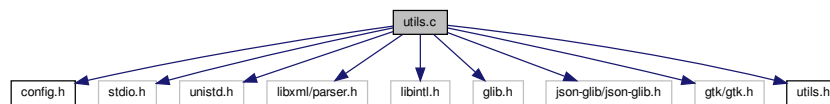
```

4.23 utils.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 "utils.h"
```

Include dependency graph for utils.c:



Functions

- void [show_message](#) (char *title, char *msg, int type)
- void [show_error](#) (char *msg)
- int [xml_node_get_int](#) (xmlNode *node, const xmlChar *prop, int *error_code)
- unsigned int [xml_node_get_uint](#) (xmlNode *node, const xmlChar *prop, int *error_code)
- unsigned int [xml_node_get_uint_with_default](#) (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)
- double [xml_node_get_float](#) (xmlNode *node, const xmlChar *prop, int *error_code)
- double [xml_node_get_float_with_default](#) (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)
- void [xml_node_set_int](#) (xmlNode *node, const xmlChar *prop, int value)
- void [xml_node_set_uint](#) (xmlNode *node, const xmlChar *prop, unsigned int value)
- void [xml_node_set_float](#) (xmlNode *node, const xmlChar *prop, double value)
- int [json_object_get_int](#) (JsonObject *object, const char *prop, int *error_code)
- unsigned int [json_object_get_uint](#) (JsonObject *object, const char *prop, int *error_code)
- unsigned int [json_object_get_uint_with_default](#) (JsonObject *object, const char *prop, unsigned int default_value, int *error_code)
- double [json_object_get_float](#) (JsonObject *object, const char *prop, int *error_code)
- double [json_object_get_float_with_default](#) (JsonObject *object, const char *prop, double default_value, int *error_code)
- void [json_object_set_int](#) (JsonObject *object, const char *prop, int value)
- void [json_object_set_uint](#) (JsonObject *object, const char *prop, unsigned int value)
- void [json_object_set_float](#) (JsonObject *object, const char *prop, double value)
- int [cores_number](#) ()
- void [process_pending](#) ()
- unsigned int [gtk_array_get_active](#) (GtkRadioButton *array[], unsigned int n)

Variables

- GtkWidget * [main_window](#)
Main GtkWidget.
- char * [error_message](#)
Error message.
- void(* [show_pending](#))() = NULL
Pointer to the function to show pending events.

4.23.1 Detailed Description

Source file to define some useful functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.23.2 Function Documentation

4.23.2.1 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line [440](#) of file [utils.c](#).

```
00441 {  
00442 #ifdef G_OS_WIN32  
00443     SYSTEM_INFO sysinfo;  
00444     GetSystemInfo (&sysinfo);  
00445     return sysinfo.dwNumberOfProcessors;  
00446 #else  
00447     return (int) sysconf (_SC_NPROCESSORS_ONLN);  
00448 #endif  
00449 }
```

4.23.2.2 gtk_array_get_active()

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

Function to get the active GtkWidget.

Returns

Active GtkWidget.

Parameters

<i>array</i>	Array of GtkRadioButtons.
<i>n</i>	Number of GtkRadioButtons.

Definition at line 469 of file [utils.c](#).

```
00471 {
00472     unsigned int i;
00473     for (i = 0; i < n; ++i)
00474         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475             break;
00476     return i;
00477 }
```

4.23.2.3 json_object_get_float()

```
double json_object_get_float (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get a floating point number of a JSON object property.

Returns

Floating point number value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Definition at line 350 of file [utils.c](#).

```
00353 {
00354     const char *buffer;
00355     double x = 0.;
00356     buffer = json_object_get_string_member (object, prop);
00357     if (!buffer)
00358         *error_code = 1;
00359     else
00360     {
00361         if (sscanf (buffer, "%lf", &x) != 1)
00362             *error_code = 2;
00363         else
00364             *error_code = 0;
00365     }
00366     return x;
00367 }
```

4.23.2.4 json_object_get_float_with_default()

```
double json_object_get_float_with_default (
    JsonObject * object,
    const char * prop,
    double default_value,
    int * error_code )
```

Function to get a floating point number of a JSON object property with a default value.

Returns

Floating point number value.

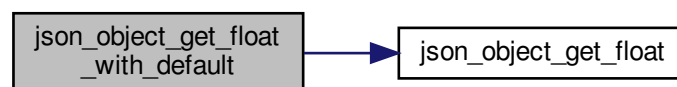
Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 376 of file [utils.c](#).

```
00382 {
00383     double x;
00384     if (json_object_get_member (object, prop))
00385         x = json_object_get_float (object, prop, error_code);
00386     else
00387     {
00388         x = default_value;
00389         *error_code = 0;
00390     }
00391     return x;
00392 }
```

Here is the call graph for this function:



4.23.2.5 json_object_get_int()

```
int json_object_get_int (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get an integer number of a JSON object property.

Returns

Integer number value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Definition at line 276 of file [utils.c](#).

```
00279 {
00280     const char *buffer;
00281     int i = 0;
00282     buffer = json_object_get_string_member (object, prop);
00283     if (!buffer)
00284         *error_code = 1;
00285     else
00286     {
00287         if (sscanf (buffer, "%d", &i) != 1)
00288             *error_code = 2;
00289         else
00290             *error_code = 0;
00291     }
00292     return i;
00293 }
```

4.23.2.6 json_object_get_uint()

```
unsigned int json_object_get_uint (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property.

Returns

Unsigned integer number value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Definition at line 301 of file [utils.c](#).

```
00304 {
00305     const char *buffer;
00306     unsigned int i = 0;
00307     buffer = json_object_get_string_member (object, prop);
00308     if (!buffer)
```

```
00309     *error_code = 1;
00310     else
00311     {
00312         if (sscanf (buffer, "%u", &i) != 1)
00313             *error_code = 2;
00314         else
00315             *error_code = 0;
00316     }
00317     return i;
00318 }
```

4.23.2.7 json_object_get_uint_with_default()

```
unsigned int json_object_get_uint_with_default (
    JsonObject * object,
    const char * prop,
    unsigned int default_value,
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property with a default value.

Returns

Unsigned integer number value.

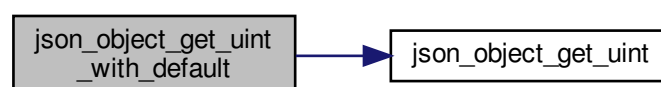
Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 327 of file [utils.c](#).

```
00332 {
00333     unsigned int i;
00334     if (json_object_get_member (object, prop))
00335         i = json_object_get_uint (object, prop, error_code);
00336     else
00337     {
00338         i = default_value;
00339         *error_code = 0;
00340     }
00341     return i;
00342 }
```

Here is the call graph for this function:



4.23.2.8 json_object_set_float()

```
void json_object_set_float (
    JsonObject * object,
    const char * prop,
    double value )
```

Function to set a floating point number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Floating point number value.

Definition at line [425](#) of file [utils.c](#).

```
00428 {
00429     char buffer[64];
00430     snprintf (buffer, 64, "%.14lg", value);
00431     json_object_set_string_member (object, prop, buffer);
00432 }
```

4.23.2.9 json_object_set_int()

```
void json_object_set_int (
    JsonObject * object,
    const char * prop,
    int value )
```

Function to set an integer number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Integer number value.

Definition at line [398](#) of file [utils.c](#).

```
00401 {
00402     char buffer[64];
00403     snprintf (buffer, 64, "%d", value);
00404     json_object_set_string_member (object, prop, buffer);
00405 }
```


4.23.2.10 json_object_set_uint()

```
void json_object_set_uint (
    JsonObject * object,
    const char * prop,
    unsigned int value )
```

Function to set an unsigned integer number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Unsigned integer number value.

Definition at line [411](#) of file [utils.c](#).

```
00415 {
00416     char buffer[64];
00417     snprintf (buffer, 64, "%u", value);
00418     json_object_set_string_member (object, prop, buffer);
00419 }
```

4.23.2.11 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line [457](#) of file [utils.c](#).

```
00458 {
00459     while (gtk_events_pending ())
00460         gtk_main_iteration ();
00461 }
```

4.23.2.12 show_error()

```
void show_error (
    char * msg )
```

Function to show a dialog with an error message.

Parameters

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

Definition at line 101 of file [utils.c](#).

```
00102 {
00103     show_message (_("ERROR!"), msg, ERROR_TYPE);
00104 }
```

Here is the call graph for this function:



4.23.2.13 show_message()

```
void show_message (
    char * title,
    char * msg,
    int type )
```

Function to show a dialog with a message.

Parameters

<i>title</i>	Title.
<i>msg</i>	Message.
<i>type</i>	Message type.

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

```
00074 {
00075     #if HAVE_GTK
00076     GtkMessageDialog *dlg;
00077
00078     // Creating the dialog
00079     dlg = (GtkMessageDialog *)
00080         gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00081                                (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00082
00083     // Setting the dialog title
00084     gtk_window_set_title (GTK_WINDOW (dlg), title);
00085
00086     // Showing the dialog and waiting response
00087     gtk_dialog_run (GTK_DIALOG (dlg));
00088
00089     // Closing and freeing memory
00090     gtk_widget_destroy (GTK_WIDGET (dlg));
00091
00092 #else
00093     printf ("%s: %s\n", title, msg);
00094 #endif
00095 }
```

4.23.2.14 xml_node_get_float()

```
double xml_node_get_float (
    xmlNode * node,
    const xmlChar * prop,
    int * error_code )
```

Function to get a floating point number of a XML node property.

Returns

Floating point number value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Definition at line 188 of file [utils.c](#).

```
00191 {
00192     double x = 0.;
00193     xmlChar *buffer;
00194     buffer = xmlGetProp (node, prop);
00195     if (!buffer)
00196         *error_code = 1;
00197     else
00198     {
00199         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00200             *error_code = 2;
00201         else
00202             *error_code = 0;
00203         xmlFree (buffer);
00204     }
00205     return x;
00206 }
```

4.23.2.15 xml_node_get_float_with_default()

```
double xml_node_get_float_with_default (
    xmlNode * node,
    const xmlChar * prop,
    double default_value,
    int * error_code )
```

Function to get a floating point number of a XML node property with a default value.

Returns

Floating point number value.

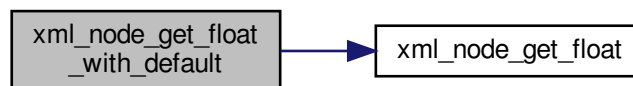
Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 215 of file [utils.c](#).

```
00219 {  
00220     double x;  
00221     if (xmlHasProp (node, prop))  
00222         x = xml_node_get_float (node, prop, error_code);  
00223     else  
00224     {  
00225         x = default_value;  
00226         *error_code = 0;  
00227     }  
00228     return x;  
00229 }
```

Here is the call graph for this function:

**4.23.2.16 xml_node_get_int()**

```
int xml_node_get_int (  
    xmlNode * node,  
    const xmlChar * prop,  
    int * error_code )
```

Function to get an integer number of a XML node property.

Returns

Integer number value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Definition at line 112 of file [utils.c](#).

```
00115 {
00116     int i = 0;
00117     xmlChar *buffer;
00118     buffer = xmlGetProp (node, prop);
00119     if (!buffer)
00120         *error_code = 1;
00121     else
00122     {
00123         if (sscanf ((char *) buffer, "%d", &i) != 1)
00124             *error_code = 2;
00125         else
00126             *error_code = 0;
00127         xmlFree (buffer);
00128     }
00129     return i;
00130 }
```

4.23.2.17 xml_node_get_uint()

```
unsigned int xml_node_get_uint (
    xmlNode * node,
    const xmlChar * prop,
    int * error_code )
```

Function to get an unsigned integer number of a XML node property.

Returns

Unsigned integer number value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Definition at line 138 of file [utils.c](#).

```
00141 {
00142     unsigned int i = 0;
00143     xmlChar *buffer;
00144     buffer = xmlGetProp (node, prop);
00145     if (!buffer)
00146         *error_code = 1;
00147     else
00148     {
00149         if (sscanf ((char *) buffer, "%u", &i) != 1)
00150             *error_code = 2;
00151         else
00152             *error_code = 0;
00153         xmlFree (buffer);
00154     }
00155     return i;
00156 }
```

4.23.2.18 xml_node_get_uint_with_default()

```
unsigned int xml_node_get_uint_with_default (
    xmlNode * node,
    const xmlChar * prop,
    unsigned int default_value,
    int * error_code )
```

Function to get an unsigned integer number of a XML node property with a default value.

Returns

Unsigned integer number value.

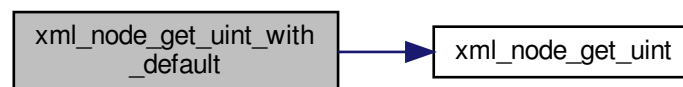
Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 165 of file [utils.c](#).

```
00170 {
00171     unsigned int i;
00172     if (xmlHasProp (node, prop))
00173         i = xml_node_get_uint (node, prop, error_code);
00174     else
00175     {
00176         i = default_value;
00177         *error_code = 0;
00178     }
00179     return i;
00180 }
```

Here is the call graph for this function:



4.23.2.19 xml_node_set_float()

```
void xml_node_set_float (
    xmlNode * node,
    const xmlChar * prop,
    double value )
```

Function to set a floating point number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Floating point number value.

Definition at line 261 of file [utils.c](#).

```
00264 {  
00265     xmlChar buffer[64];  
00266     snprintf ((char *) buffer, 64, "%.14lg", value);  
00267     xmlSetProp (node, prop, buffer);  
00268 }
```

4.23.2.20 xml_node_set_int()

```
void xml_node_set_int (  
    xmlNode * node,  
    const xmlChar * prop,  
    int value )
```

Function to set an integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Integer number value.

Definition at line 235 of file [utils.c](#).

```
00238 {  
00239     xmlChar buffer[64];  
00240     snprintf ((char *) buffer, 64, "%d", value);  
00241     xmlSetProp (node, prop, buffer);  
00242 }
```

4.23.2.21 xml_node_set_uint()

```
void xml_node_set_uint (  
    xmlNode * node,  
    const xmlChar * prop,  
    unsigned int value )
```

Function to set an unsigned integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Unsigned integer number value.

Definition at line 248 of file [utils.c](#).

```

00251 {
00252     xmlChar buffer[64];
00253     snprintf ((char *) buffer, 64, "%u", value);
00254     xmlSetProp (node, prop, buffer);
00255 }
```

4.24 utils.c

```

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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <unistd.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <json-glib/json-glib.h>
00040 #ifdef G_OS_WIN32
00041 #include <windows.h>
00042 #endif
00043 #if HAVE_GTK
00044 #include <gtk/gtk.h>
00045 #endif
00046 #include "utils.h"
00047
00048 #if HAVE_GTK
00049 GtkWidget *main_window;
00050 #endif
00051
00052 char *error_message;
00053 void (*show_pending) () = NULL;
00054
00055 void
00056 show_message (char *title,
00057              char *msg,
00058              int type
00059 #if !HAVE_GTK
00060 )

```



```

00070         __attribute__ ((unused))
00071 #endif
00072     )
00073 {
00074     #if HAVE_GTK
00075     GtkWidgetDialog *dlg;
00076     // Creating the dialog
00077     dlg = (GtkMessageDialog *)
00078         gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00079                                (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00080     // Setting the dialog title
00081     gtk_window_set_title (GTK_WINDOW (dlg), title);
00082     // Showing the dialog and waiting response
00083     gtk_dialog_run (GTK_DIALOG (dlg));
00084     // Closing and freeing memory
00085     gtk_widget_destroy (GTK_WIDGET (dlg));
00086 #else
00087     printf ("%s: %s\n", title, msg);
00088 #endif
00089 }
00090 void
00091 show_error (char *msg)
00092 {
00093     show_message (_("ERROR!"), msg, ERROR_TYPE);
00094 }
00095 int
00096 xml_node_get_int (xmlNode * node,
00097                  const xmlChar * prop,
00098                  int *error_code)
00099 {
00100     int i = 0;
00101     xmlChar *buffer;
00102     buffer = xmlGetProp (node, prop);
00103     if (!buffer)
00104         *error_code = 1;
00105     else
00106     {
00107         if (sscanf ((char *) buffer, "%d", &i) != 1)
00108             *error_code = 2;
00109         else
00110             *error_code = 0;
00111         xmlFree (buffer);
00112     }
00113     return i;
00114 }
00115 unsigned int
00116 xml_node_get_uint (xmlNode * node,
00117                   const xmlChar * prop,
00118                   int *error_code)
00119 {
00120     unsigned int i = 0;
00121     xmlChar *buffer;
00122     buffer = xmlGetProp (node, prop);
00123     if (!buffer)
00124         *error_code = 1;
00125     else
00126     {
00127         if (sscanf ((char *) buffer, "%u", &i) != 1)
00128             *error_code = 2;
00129         else
00130             *error_code = 0;
00131         xmlFree (buffer);
00132     }
00133     return i;
00134 }
00135 unsigned int
00136 xml_node_get_uint_with_default (xmlNode * node,
00137                                const xmlChar * prop,
00138                                unsigned int default_value,
00139                                int *error_code)
00140 {
00141     unsigned int i;
00142     if (xmlHasProp (node, prop))
00143         i = xml_node_get_uint (node, prop, error_code);
00144     else
00145     {
00146         i = default_value;
00147         *error_code = 0;
00148     }
00149 }

```

```

00178     }
00179     return i;
00180 }
00181
00182 double
00183 xml_node_get_float (xmlNode * node,
00184                     const xmlChar * prop,
00185                     int *error_code)
00186 {
00187     double x = 0.;
00188     xmlChar *buffer;
00189     buffer = xmlGetProp (node, prop);
00190     if (!buffer)
00191         *error_code = 1;
00192     else
00193     {
00194         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00195             *error_code = 2;
00196         else
00197             *error_code = 0;
00198         xmlFree (buffer);
00199     }
00200     return x;
00201 }
00202
00203 double
00204 xml_node_get_float_with_default (xmlNode * node,
00205                                 const xmlChar * prop,
00206                                 double default_value,
00207                                 int *error_code)
00208 {
00209     double x;
00210     if (xmlHasProp (node, prop))
00211         x = xml_node_get_float (node, prop, error_code);
00212     else
00213     {
00214         x = default_value;
00215         *error_code = 0;
00216     }
00217     return x;
00218 }
00219
00220 void
00221 xml_node_set_int (xmlNode * node,
00222                  const xmlChar * prop,
00223                  int value)
00224 {
00225     xmlChar buffer[64];
00226     snprintf ((char *) buffer, 64, "%d", value);
00227     xmlSetProp (node, prop, buffer);
00228 }
00229
00230 void
00231 xml_node_set_uint (xmlNode * node,
00232                   const xmlChar * prop,
00233                   unsigned int value)
00234 {
00235     xmlChar buffer[64];
00236     snprintf ((char *) buffer, 64, "%u", value);
00237     xmlSetProp (node, prop, buffer);
00238 }
00239
00240 void
00241 xml_node_set_float (xmlNode * node,
00242                    const xmlChar * prop,
00243                    double value)
00244 {
00245     xmlChar buffer[64];
00246     snprintf ((char *) buffer, 64, "%.14lg", value);
00247     xmlSetProp (node, prop, buffer);
00248 }
00249
00250 int
00251 json_object_get_int (JsonObject * object,
00252                     const char *prop,
00253                     int *error_code)
00254 {
00255     const char *buffer;
00256     int i = 0;
00257     buffer = json_object_get_string_member (object, prop);
00258     if (!buffer)
00259         *error_code = 1;
00260     else
00261     {
00262         if (sscanf (buffer, "%d", &i) != 1)
00263             *error_code = 2;
00264         else

```

```
00290         *error_code = 0;
00291     }
00292     return i;
00293 }
00294
00300 unsigned int
00301 json_object_get_uint (JsonObject * object,
00302                      const char *prop,
00303                      int *error_code)
00304 {
00305     const char *buffer;
00306     unsigned int i = 0;
00307     buffer = json_object_get_string_member (object, prop);
00308     if (!buffer)
00309         *error_code = 1;
00310     else
00311     {
00312         if (sscanf (buffer, "%u", &i) != 1)
00313             *error_code = 2;
00314         else
00315             *error_code = 0;
00316     }
00317     return i;
00318 }
00319
00326 unsigned int
00327 json_object_get_uint_with_default (JsonObject * object,
00328                                   const char *prop,
00329                                   unsigned int default_value,
00330                                   int *error_code)
00331 {
00332     {
00333         unsigned int i;
00334         if (json_object_get_member (object, prop))
00335             i = json_object_get_uint (object, prop, error_code);
00336         else
00337         {
00338             i = default_value;
00339             *error_code = 0;
00340         }
00341         return i;
00342     }
00343 }
00344
00349 double
00350 json_object_get_float (JsonObject * object,
00351                       const char *prop,
00352                       int *error_code)
00353 {
00354     const char *buffer;
00355     double x = 0.;
00356     buffer = json_object_get_string_member (object, prop);
00357     if (!buffer)
00358         *error_code = 1;
00359     else
00360     {
00361         if (sscanf (buffer, "%lf", &x) != 1)
00362             *error_code = 2;
00363         else
00364             *error_code = 0;
00365     }
00366     return x;
00367 }
00368
00375 double
00376 json_object_get_float_with_default (JsonObject * object,
00377                                    const char *prop,
00378                                    double default_value,
00379                                    int *error_code)
00380 {
00381     {
00382         double x;
00383         if (json_object_get_member (object, prop))
00384             x = json_object_get_float (object, prop, error_code);
00385         else
00386         {
00387             x = default_value;
00388             *error_code = 0;
00389         }
00390         return x;
00391     }
00392 }
00393
00397 void
00398 json_object_set_int (JsonObject * object,
00399                     const char *prop,
00400                     int value)
00401 {
00402     char buffer[64];
00403     snprintf (buffer, 64, "%d", value);
00404     json_object_set_string_member (object, prop, buffer);
00405 }
```

```

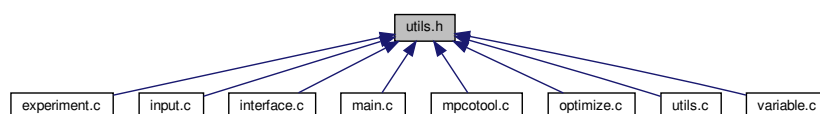
00405 }
00406
00410 void
00411 json_object_set_uint (JsonObject * object,
00412                      const char *prop,
00413                      unsigned int value)
00414 {
00415     char buffer[64];
00416     snprintf (buffer, 64, "%u", value);
00417     json_object_set_string_member (object, prop, buffer);
00418 }
00419
00420
00424 void
00425 json_object_set_float (JsonObject * object,
00426                      const char *prop,
00427                      double value)
00428 {
00429     char buffer[64];
00430     snprintf (buffer, 64, "%.14lg", value);
00431     json_object_set_string_member (object, prop, buffer);
00432 }
00433
00439 int
00440 cores_number ()
00441 {
00442     #ifdef G_OS_WIN32
00443         SYSTEM_INFO sysinfo;
00444         GetSystemInfo (&sysinfo);
00445         return sysinfo.dwNumberOfProcessors;
00446     #else
00447         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00448     #endif
00449 }
00450
00451 #if HAVE_GTK
00452
00456 void
00457 process_pending ()
00458 {
00459     while (gtk_events_pending ())
00460         gtk_main_iteration ();
00461 }
00462
00468 unsigned int
00469 gtk_array_get_active (GtkRadioButton * array[],
00470                     unsigned int n)
00471 {
00472     unsigned int i;
00473     for (i = 0; i < n; ++i)
00474         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475             break;
00476     return i;
00477 }
00478
00479 #endif

```

4.25 utils.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.

Functions

- void [show_message](#) (char *title, char *msg, int type)
- void [show_error](#) (char *msg)
- int [xml_node_get_int](#) (xmlNode *node, const xmlChar *prop, int *error_code)
- unsigned int [xml_node_get_uint](#) (xmlNode *node, const xmlChar *prop, int *error_code)
- unsigned int [xml_node_get_uint_with_default](#) (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)
- double [xml_node_get_float](#) (xmlNode *node, const xmlChar *prop, int *error_code)
- double [xml_node_get_float_with_default](#) (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)
- void [xml_node_set_int](#) (xmlNode *node, const xmlChar *prop, int value)
- void [xml_node_set_uint](#) (xmlNode *node, const xmlChar *prop, unsigned int value)
- void [xml_node_set_float](#) (xmlNode *node, const xmlChar *prop, double value)
- int [json_object_get_int](#) (JsonObject *object, const char *prop, int *error_code)
- unsigned int [json_object_get_uint](#) (JsonObject *object, const char *prop, int *error_code)
- unsigned int [json_object_get_uint_with_default](#) (JsonObject *object, const char *prop, unsigned int default_value, int *error_code)
- double [json_object_get_float](#) (JsonObject *object, const char *prop, int *error_code)
- double [json_object_get_float_with_default](#) (JsonObject *object, const char *prop, double default_value, int *error_code)
- void [json_object_set_int](#) (JsonObject *object, const char *prop, int value)
- void [json_object_set_uint](#) (JsonObject *object, const char *prop, unsigned int value)
- void [json_object_set_float](#) (JsonObject *object, const char *prop, double value)
- int [cores_number](#) ()
- void [process_pending](#) ()
- unsigned int [gtk_array_get_active](#) (GtkRadioButton *array[], unsigned int n)

Variables

- GtkWidget * [main_window](#)
Main GtkWidget.
- char * [error_message](#)
Error message.
- void(* [show_pending](#))()
Pointer to the function to show pending events.

4.25.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.25.2 Function Documentation

4.25.2.1 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line [440](#) of file [utils.c](#).

```
00441 {
00442     #ifdef G_OS_WIN32
00443         SYSTEM_INFO sysinfo;
00444         GetSystemInfo (&sysinfo);
00445         return sysinfo.dwNumberOfProcessors;
00446     #else
00447         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00448     #endif
00449 }
```

4.25.2.2 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

<i>array</i>	Array of GtkRadioButtons.
<i>n</i>	Number of GtkRadioButtons.

Definition at line [469](#) of file [utils.c](#).

```
00471 {
00472     unsigned int i;
00473     for (i = 0; i < n; ++i)
00474         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475             break;
00476     return i;
00477 }
```

4.25.2.3 json_object_get_float()

```
double json_object_get_float (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get a floating point number of a JSON object property.

Returns

Floating point number value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Definition at line 350 of file [utils.c](#).

```
00353 {
00354     const char *buffer;
00355     double x = 0.;
00356     buffer = json_object_get_string_member (object, prop);
00357     if (!buffer)
00358         *error_code = 1;
00359     else
00360     {
00361         if (sscanf (buffer, "%lf", &x) != 1)
00362             *error_code = 2;
00363         else
00364             *error_code = 0;
00365     }
00366     return x;
00367 }
```

4.25.2.4 json_object_get_float_with_default()

```
double json_object_get_float_with_default (
    JsonObject * object,
    const char * prop,
    double default_value,
    int * error_code )
```

Function to get a floating point number of a JSON object property with a default value.

Returns

Floating point number value.

Parameters

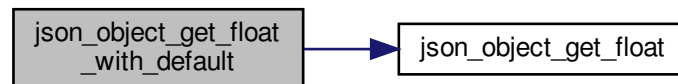
<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 376 of file [utils.c](#).

```

00382 {
00383     double x;
00384     if (json_object_get_member (object, prop))
00385         x = json_object_get_float (object, prop, error_code);
00386     else
00387     {
00388         x = default_value;
00389         *error_code = 0;
00390     }
00391     return x;
00392 }
```

Here is the call graph for this function:

**4.25.2.5 json_object_get_int()**

```

int json_object_get_int (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get an integer number of a JSON object property.

Returns

Integer number value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Definition at line 276 of file [utils.c](#).

```
00279 {
00280     const char *buffer;
00281     int i = 0;
00282     buffer = json_object_get_string_member (object, prop);
00283     if (!buffer)
00284         *error_code = 1;
00285     else
00286     {
00287         if (sscanf (buffer, "%d", &i) != 1)
00288             *error_code = 2;
00289         else
00290             *error_code = 0;
00291     }
00292     return i;
00293 }
```

4.25.2.6 json_object_get_uint()

```
unsigned int json_object_get_uint (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property.

Returns

Unsigned integer number value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Definition at line 301 of file [utils.c](#).

```
00304 {
00305     const char *buffer;
00306     unsigned int i = 0;
00307     buffer = json_object_get_string_member (object, prop);
00308     if (!buffer)
00309         *error_code = 1;
00310     else
00311     {
00312         if (sscanf (buffer, "%u", &i) != 1)
00313             *error_code = 2;
00314         else
00315             *error_code = 0;
00316     }
00317     return i;
00318 }
```

4.25.2.7 json_object_get_uint_with_default()

```
unsigned int json_object_get_uint_with_default (  
    JsonObject * object,  
    const char * prop,  
    unsigned int default_value,  
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property with a default value.

Returns

Unsigned integer number value.

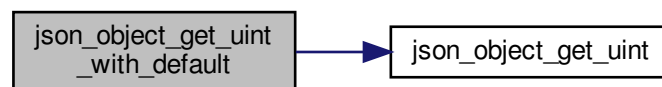
Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line [327](#) of file [utils.c](#).

```
00332 {  
00333     unsigned int i;  
00334     if (json_object_get_member (object, prop))  
00335         i = json_object_get_uint (object, prop, error_code);  
00336     else  
00337     {  
00338         i = default_value;  
00339         *error_code = 0;  
00340     }  
00341     return i;  
00342 }
```

Here is the call graph for this function:



4.25.2.8 json_object_set_float()

```
void json_object_set_float (  
    JsonObject * object,  
    const char * prop,  
    double value )
```

Function to set a floating point number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Floating point number value.

Definition at line 425 of file [utils.c](#).

```
00428 {  
00429     char buffer[64];  
00430     snprintf (buffer, 64, "%.14lg", value);  
00431     json_object_set_string_member (object, prop, buffer);  
00432 }
```

4.25.2.9 json_object_set_int()

```
void json_object_set_int (  
    JsonObject * object,  
    const char * prop,  
    int value )
```

Function to set an integer number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Integer number value.

Definition at line 398 of file [utils.c](#).

```
00401 {  
00402     char buffer[64];  
00403     snprintf (buffer, 64, "%d", value);  
00404     json_object_set_string_member (object, prop, buffer);  
00405 }
```

4.25.2.10 json_object_set_uint()

```
void json_object_set_uint (  
    JsonObject * object,  
    const char * prop,  
    unsigned int value )
```

Function to set an unsigned integer number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Unsigned integer number value.

Definition at line 411 of file [utils.c](#).

```
00415 {  
00416     char buffer[64];  
00417     snprintf (buffer, 64, "%u", value);  
00418     json_object_set_string_member (object, prop, buffer);  
00419 }
```

4.25.2.11 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 457 of file [utils.c](#).

```
00458 {  
00459     while (gtk_events_pending ())  
00460         gtk_main_iteration ();  
00461 }
```

4.25.2.12 show_error()

```
void show_error (  
    char * msg )
```

Function to show a dialog with an error message.

Parameters

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

Definition at line 101 of file [utils.c](#).

```
00102 {  
00103     show_message (_("ERROR!"), msg, ERROR_TYPE);  
00104 }
```

Here is the call graph for this function:



4.25.2.13 show_message()

```
void show_message (  
    char * title,  
    char * msg,  
    int type )
```

Function to show a dialog with a message.

Parameters

<i>title</i>	Title.
<i>msg</i>	Message.
<i>type</i>	Message type.

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

```
00074 {  
00075 #if HAVE_GTK  
00076     GtkMessageDialog *dlg;  
00077  
00078     // Creating the dialog  
00079     dlg = (GtkMessageDialog *)  
00080         gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,  
00081                                 (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);  
00082  
00083     // Setting the dialog title  
00084     gtk_window_set_title (GTK_WINDOW (dlg), title);  
00085  
00086     // Showing the dialog and waiting response  
00087     gtk_dialog_run (GTK_DIALOG (dlg));  
00088  
00089     // Closing and freeing memory  
00090     gtk_widget_destroy (GTK_WIDGET (dlg));  
00091  
00092 #else  
00093     printf ("%s: %s\n", title, msg);  
00094 #endif  
00095 }
```

4.25.2.14 xml_node_get_float()

```
double xml_node_get_float (
    xmlNode * node,
    const xmlChar * prop,
    int * error_code )
```

Function to get a floating point number of a XML node property.

Returns

Floating point number value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Definition at line 188 of file [utils.c](#).

```
00191 {
00192     double x = 0.;
00193     xmlChar *buffer;
00194     buffer = xmlGetProp (node, prop);
00195     if (!buffer)
00196         *error_code = 1;
00197     else
00198     {
00199         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00200             *error_code = 2;
00201         else
00202             *error_code = 0;
00203         xmlFree (buffer);
00204     }
00205     return x;
00206 }
```

4.25.2.15 xml_node_get_float_with_default()

```
double xml_node_get_float_with_default (
    xmlNode * node,
    const xmlChar * prop,
    double default_value,
    int * error_code )
```

Function to get a floating point number of a XML node property with a default value.

Returns

Floating point number value.

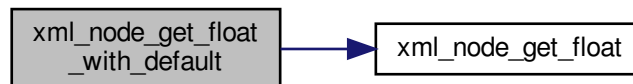
Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 215 of file [utils.c](#).

```
00219 {  
00220     double x;  
00221     if (xmlHasProp (node, prop))  
00222         x = xml_node_get_float (node, prop, error_code);  
00223     else  
00224     {  
00225         x = default_value;  
00226         *error_code = 0;  
00227     }  
00228     return x;  
00229 }
```

Here is the call graph for this function:



4.25.2.16 xml_node_get_int()

```
int xml_node_get_int (  
    xmlNode * node,  
    const xmlChar * prop,  
    int * error_code )
```

Function to get an integer number of a XML node property.

Returns

Integer number value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Definition at line 112 of file [utils.c](#).

```
00115 {
00116     int i = 0;
00117     xmlChar *buffer;
00118     buffer = xmlGetProp (node, prop);
00119     if (!buffer)
00120         *error_code = 1;
00121     else
00122     {
00123         if (sscanf ((char *) buffer, "%d", &i) != 1)
00124             *error_code = 2;
00125         else
00126             *error_code = 0;
00127         xmlFree (buffer);
00128     }
00129     return i;
00130 }
```

4.25.2.17 xml_node_get_uint()

```
unsigned int xml_node_get_uint (
    xmlNode * node,
    const xmlChar * prop,
    int * error_code )
```

Function to get an unsigned integer number of a XML node property.

Returns

Unsigned integer number value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Definition at line 138 of file [utils.c](#).

```
00141 {
00142     unsigned int i = 0;
00143     xmlChar *buffer;
00144     buffer = xmlGetProp (node, prop);
00145     if (!buffer)
00146         *error_code = 1;
00147     else
00148     {
00149         if (sscanf ((char *) buffer, "%u", &i) != 1)
00150             *error_code = 2;
00151         else
00152             *error_code = 0;
00153         xmlFree (buffer);
00154     }
00155     return i;
00156 }
```


4.25.2.18 xml_node_get_uint_with_default()

```
unsigned int xml_node_get_uint_with_default (
    xmlNode * node,
    const xmlChar * prop,
    unsigned int default_value,
    int * error_code )
```

Function to get an unsigned integer number of a XML node property with a default value.

Returns

Unsigned integer number value.

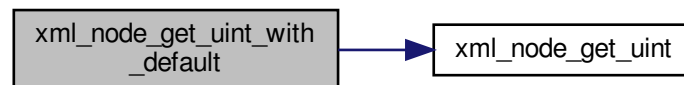
Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Definition at line 165 of file [utils.c](#).

```
00170 {
00171     unsigned int i;
00172     if (xmlHasProp (node, prop))
00173         i = xml_node_get_uint (node, prop, error_code);
00174     else
00175     {
00176         i = default_value;
00177         *error_code = 0;
00178     }
00179     return i;
00180 }
```

Here is the call graph for this function:



4.25.2.19 xml_node_set_float()

```
void xml_node_set_float (
    xmlNode * node,
    const xmlChar * prop,
    double value )
```

Function to set a floating point number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Floating point number value.

Definition at line 261 of file [utils.c](#).

```
00264 {  
00265     xmlChar buffer[64];  
00266     snprintf ((char *) buffer, 64, "%.14lg", value);  
00267     xmlSetProp (node, prop, buffer);  
00268 }
```

4.25.2.20 xml_node_set_int()

```
void xml_node_set_int (  
    xmlNode * node,  
    const xmlChar * prop,  
    int value )
```

Function to set an integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Integer number value.

Definition at line 235 of file [utils.c](#).

```
00238 {  
00239     xmlChar buffer[64];  
00240     snprintf ((char *) buffer, 64, "%d", value);  
00241     xmlSetProp (node, prop, buffer);  
00242 }
```

4.25.2.21 xml_node_set_uint()

```
void xml_node_set_uint (  
    xmlNode * node,  
    const xmlChar * prop,  
    unsigned int value )
```

Function to set an unsigned integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Unsigned integer number value.

Definition at line 248 of file [utils.c](#).

```
00251 {
00252     xmlChar buffer[64];
00253     snprintf ((char *) buffer, 64, "%u", value);
00254     xmlSetProp (node, prop, buffer);
00255 }
```

4.26 utils.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013 1. Redistributions of source code must retain the above copyright notice,
00014    this list of conditions and the following disclaimer.
00015
00016 2. Redistributions in binary form must reproduce the above copyright notice,
00017    this list of conditions and the following disclaimer in the
00018    documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef UTILS__H
00033 #define UTILS__H 1
00034
00035 #if HAVE_GTK
00036 #define ERROR_TYPE GTK_MESSAGE_ERROR
00037 #define INFO_TYPE GTK_MESSAGE_INFO
00038 extern GtkWidget *main_window;
00039 #else
00040 #define ERROR_TYPE 0
00041 #define INFO_TYPE 0
00042 #endif
00043
00044 extern char *error_message;
00045 extern void (*show_pending) ();
00046
00047 // Public functions
00048 void show_message (char *title, char *msg, int type);
00049 void show_error (char *msg);
00050 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00051 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00052                                int *error_code);
00053 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00054                                              const xmlChar * prop,
00055                                              unsigned int default_value,
00056                                              int *error_code);
00057 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00058                            int *error_code);
00059 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
```

```

00072     double default_value, int *error_code);
00073 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00074 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00075     unsigned int value);
00076 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00077 int json_object_get_int (JsonObject * object, const char *prop,
00078     int *error_code);
00079 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00080     int *error_code);
00081 unsigned int json_object_get_uint_with_default (JsonObject * object,
00082     const char *prop,
00083     unsigned int default_value,
00084     int *error_code);
00085 double json_object_get_float (JsonObject * object, const char *prop,
00086     int *error_code);
00087 double json_object_get_float_with_default (JsonObject * object,
00088     const char *prop,
00089     double default_value,
00090     int *error_code);
00091 void json_object_set_int (JsonObject * object, const char *prop, int value);
00092 void json_object_set_uint (JsonObject * object, const char *prop,
00093     unsigned int value);
00094 void json_object_set_float (JsonObject * object, const char *prop,
00095     double value);
00096 int cores_number ();
00097 #if HAVE_GTK
00098 void process_pending ();
00099 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00100 #endif
00101
00102 #endif

```

4.27 variable.c File Reference

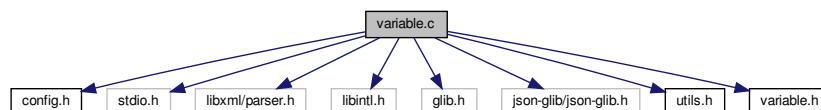
Source file to define the variable data.

```

#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "variable.h"

```

Include dependency graph for variable.c:



Macros

- `#define DEBUG_VARIABLE 0`
Macro to debug variable functions.

Functions

- void [variable_new](#) ([Variable](#) *variable)
- void [variable_free](#) ([Variable](#) *variable, unsigned int type)
- void [variable_error](#) ([Variable](#) *variable, char *message)
- int [variable_open_xml](#) ([Variable](#) *variable, xmlDoc *node, unsigned int algorithm, unsigned int nsteps)
- int [variable_open_json](#) ([Variable](#) *variable, cJSON *node, unsigned int algorithm, unsigned int nsteps)

Variables

- const char * [format](#) [[NPRECISIONS](#)]
Array of C-strings with variable formats.
- const double [precision](#) [[NPRECISIONS](#)]
Array of variable precisions.

4.27.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.27.2 Function Documentation

4.27.2.1 [variable_error\(\)](#)

```
void variable_error (  
    Variable * variable,  
    char * message )
```

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

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	Error message.

Definition at line [100](#) of file [variable.c](#).

```

00104 {
00105     char buffer[64];
00106     if (!variable->name)
00107         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00108     else
00109         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00110     error_message = g_strdup (buffer);
00111 }

```

4.27.2.2 variable_free()

```

void variable_free (
    Variable * variable,
    unsigned int type )

```

Function to free the memory of a [Variable](#) struct.

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	Type of input file.

Definition at line 79 of file [variable.c](#).

```

00083 {
00084     #if DEBUG_VARIABLE
00085         fprintf (stderr, "variable_free: start\n");
00086     #endif
00087     if (type == INPUT_TYPE_XML)
00088         xmlFree (variable->name);
00089     else
00090         g_free (variable->name);
00091     #if DEBUG_VARIABLE
00092         fprintf (stderr, "variable_free: end\n");
00093     #endif
00094 }

```

4.27.2.3 variable_new()

```

void variable_new (
    Variable * variable )

```

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

Parameters

<i>variable</i>	Variable struct.
-----------------	----------------------------------

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

```

00065 {

```

```

00066 #if DEBUG_VARIABLE
00067     fprintf (stderr, "variable_new: start\n");
00068 #endif
00069     variable->name = NULL;
00070 #if DEBUG_VARIABLE
00071     fprintf (stderr, "variable_new: end\n");
00072 #endif
00073 }

```

4.27.2.4 variable_open_json()

```

int variable_open_json (
    Variable * variable,
    XmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 279 of file [variable.c](#).

```

00284 {
00285     JsonObject *object;
00286     const char *label;
00287     int error_code;
00288 #if DEBUG_VARIABLE
00289     fprintf (stderr, "variable_open_json: start\n");
00290 #endif
00291     object = json_node_get_object (node);
00292     label = json_object_get_string_member (object, LABEL_NAME);
00293     if (!label)
00294     {
00295         variable_error (variable, _("no name"));
00296         goto exit_on_error;
00297     }
00298     variable->name = g_strdup (label);
00299     if (json_object_get_member (object, LABEL_MINIMUM))
00300     {
00301         variable->rangemin
00302         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00303         if (error_code)
00304         {
00305             variable_error (variable, _("bad minimum"));
00306             goto exit_on_error;
00307         }
00308         variable->rangeminabs
00309         = json_object_get_float_with_default (object,
00310             LABEL_ABSOLUTE_MINIMUM,
00311             -G_MAXDOUBLE, &error_code);
00312         if (error_code)
00313         {
00314             variable_error (variable, _("bad absolute minimum"));

```

```

00314         goto exit_on_error;
00315     }
00316     if (variable->rangemin < variable->rangeminabs)
00317     {
00318         variable_error (variable, _("minimum range not allowed"));
00319         goto exit_on_error;
00320     }
00321 }
00322 else
00323 {
00324     variable_error (variable, _("no minimum range"));
00325     goto exit_on_error;
00326 }
00327 if (json_object_get_member (object, LABEL_MAXIMUM))
00328 {
00329     variable->rangemax
00330     = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00331     if (error_code)
00332     {
00333         variable_error (variable, _("bad maximum"));
00334         goto exit_on_error;
00335     }
00336     variable->rangemaxabs
00337     = json_object_get_float_with_default (object,
00338     LABEL_ABSOLUTE_MAXIMUM,
00339     G_MAXDOUBLE, &error_code);
00340     if (error_code)
00341     {
00342         variable_error (variable, _("bad absolute maximum"));
00343         goto exit_on_error;
00344     }
00345     if (variable->rangemax > variable->rangemaxabs)
00346     {
00347         variable_error (variable, _("maximum range not allowed"));
00348         goto exit_on_error;
00349     }
00350     if (variable->rangemax < variable->rangemin)
00351     {
00352         variable_error (variable, _("bad range"));
00353         goto exit_on_error;
00354     }
00355 }
00356 else
00357 {
00358     variable_error (variable, _("no maximum range"));
00359     goto exit_on_error;
00360 }
00361 variable->precision
00362 = json_object_get_uint_with_default (object,
00363 LABEL_PRECISION,
00364 DEFAULT_PRECISION, &error_code);
00365 if (error_code || variable->precision >= NPRECISIONS)
00366 {
00367     variable_error (variable, _("bad precision"));
00368     goto exit_on_error;
00369 }
00370 if (algorithm == ALGORITHM_SWEEP || algorithm ==
00371 ALGORITHM_ORTHOGONAL)
00372 {
00373     if (json_object_get_member (object, LABEL_NSWEEPS))
00374     {
00375         variable->nsweeps
00376         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00377         if (error_code || !variable->nsweeps)
00378         {
00379             variable_error (variable, _("bad sweeps"));
00380             goto exit_on_error;
00381         }
00382     }
00383 }
00384 else
00385 {
00386     variable_error (variable, _("no sweeps number"));
00387     goto exit_on_error;
00388 }
00389 #if DEBUG_VARIABLE
00390 fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00391 #endif
00392 if (algorithm == ALGORITHM_GENETIC)
00393 {
00394     // Obtaining bits representing each variable
00395     if (json_object_get_member (object, LABEL_NBITS))
00396     {
00397         variable->nbits
00398         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00399         if (error_code || !variable->nbits)
00400         {

```



```

00398         variable_error (variable, _("invalid bits number"));
00399         goto exit_on_error;
00400     }
00401 }
00402 else
00403 {
00404     variable_error (variable, _("no bits number"));
00405     goto exit_on_error;
00406 }
00407 }
00408 else if (nsteps)
00409 {
00410     variable->step = json_object_get_float (object,
00411     LABEL_STEP, &error_code);
00412     if (error_code || variable->step < 0.)
00413     {
00414         variable_error (variable, _("bad step size"));
00415         goto exit_on_error;
00416     }
00417 }
00418 #if DEBUG_VARIABLE
00419 fprintf (stderr, "variable_open_json: end\n");
00420 #endif
00421 return 1;
00422 exit_on_error:
00423 variable_free (variable, INPUT_TYPE_JSON);
00424 #if DEBUG_VARIABLE
00425 fprintf (stderr, "variable_open_json: end\n");
00426 #endif
00427 return 0;
00428 }

```

Here is the call graph for this function:



4.27.2.5 variable_open_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 119 of file [variable.c](#).

```

00124 {
00125     int error_code;
00126
00127     #if DEBUG_VARIABLE
00128         fprintf (stderr, "variable_open_xml: start\n");
00129     #endif
00130
00131     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00132     if (!variable->name)
00133     {
00134         variable_error (variable, _("no name"));
00135         goto exit_on_error;
00136     }
00137     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00138     {
00139         variable->rangemin
00140             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00141                                   &error_code);
00142         if (error_code)
00143         {
00144             variable_error (variable, _("bad minimum"));
00145             goto exit_on_error;
00146         }
00147         variable->rangeminabs = xml_node_get_float_with_default
00148             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00149             &error_code);
00150         if (error_code)
00151         {
00152             variable_error (variable, _("bad absolute minimum"));
00153             goto exit_on_error;
00154         }
00155         if (variable->rangemin < variable->rangeminabs)
00156         {
00157             variable_error (variable, _("minimum range not allowed"));
00158             goto exit_on_error;
00159         }
00160     }
00161     else
00162     {
00163         variable_error (variable, _("no minimum range"));
00164         goto exit_on_error;
00165     }
00166     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00167     {
00168         variable->rangemax
00169             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00170                                   &error_code);
00171         if (error_code)
00172         {
00173             variable_error (variable, _("bad maximum"));
00174             goto exit_on_error;
00175         }
00176         variable->rangemaxabs = xml_node_get_float_with_default
00177             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00178             &error_code);
00179         if (error_code)
00180         {
00181             variable_error (variable, _("bad absolute maximum"));
00182             goto exit_on_error;
00183         }
00184         if (variable->rangemax > variable->rangemaxabs)
00185         {
00186             variable_error (variable, _("maximum range not allowed"));
00187             goto exit_on_error;
00188         }
00189         if (variable->rangemax < variable->rangemin)
00190         {
00191             variable_error (variable, _("bad range"));
00192             goto exit_on_error;
00193         }
00194     }
00195     else
00196     {
00197         variable_error (variable, _("no maximum range"));
00198         goto exit_on_error;
00199     }
00200     variable->precision
00201         = xml_node_get_uint_with_default (node, (const xmlChar *)
00202         LABEL_PRECISION,
00203         DEFAULT_PRECISION, &error_code);
00204     if (error_code || variable->precision >= NPRECISIONS)
00205     {
00206         variable_error (variable, _("bad precision"));
00207         goto exit_on_error;

```

```

00207     }
00208     if (algorithm == ALGORITHM_SWEEP || algorithm ==
        ALGORITHM_ORTHOGONAL)
00209     {
00210         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00211         {
00212             variable->nsweeps
00213             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00214                                 &error_code);
00215             if (error_code || !variable->nsweeps)
00216             {
00217                 variable_error (variable, _("bad sweeps"));
00218                 goto exit_on_error;
00219             }
00220         }
00221         else
00222         {
00223             variable_error (variable, _("no sweeps number"));
00224             goto exit_on_error;
00225         }
00226         #if DEBUG_VARIABLE
00227             fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00228         #endif
00229     }
00230     if (algorithm == ALGORITHM_GENETIC)
00231     {
00232         // Obtaining bits representing each variable
00233         if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00234         {
00235             variable->nbits
00236             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00237                                 &error_code);
00238             if (error_code || !variable->nbits)
00239             {
00240                 variable_error (variable, _("invalid bits number"));
00241                 goto exit_on_error;
00242             }
00243         }
00244         else
00245         {
00246             variable_error (variable, _("no bits number"));
00247             goto exit_on_error;
00248         }
00249     }
00250     else if (nsteps)
00251     {
00252         variable->step
00253         = xml_node_get_float (node, (const xmlChar *)
        LABEL_STEP, &error_code);
00254         if (error_code || variable->step < 0.)
00255         {
00256             variable_error (variable, _("bad step size"));
00257             goto exit_on_error;
00258         }
00259     }
00260     #if DEBUG_VARIABLE
00261         fprintf (stderr, "variable_open_xml: end\n");
00262     #endif
00263     return 1;
00264 exit_on_error:
00265     variable_free (variable, INPUT_TYPE_XML);
00266     #if DEBUG_VARIABLE
00267         fprintf (stderr, "variable_open_xml: end\n");
00268     #endif
00269     return 0;
00270 }
00271 }

```

Here is the call graph for this function:



4.27.3 Variable Documentation

4.27.3.1 format

```
const char* format[NPRECISIONS]
```

Initial value:

```
= {
    "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
    "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
}
```

Array of C-strings with variable formats.

Definition at line 50 of file [variable.c](#).

4.27.3.2 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 55 of file [variable.c](#).

4.28 variable.c

```
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 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```

00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "variable.h"
00041
00042 #define DEBUG_VARIABLE 0
00043
00044 const char *format[NPRECISIONS] = {
00045     "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
00046     "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
00047 };
00048
00049 const double precision[NPRECISIONS] = {
00050     1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00051     1e-12, 1e-13, 1e-14
00052 };
00053
00054 void
00055 variable_new (Variable * variable)
00056 {
00057     #if DEBUG_VARIABLE
00058         fprintf (stderr, "variable_new: start\n");
00059     #endif
00060     variable->name = NULL;
00061     #if DEBUG_VARIABLE
00062         fprintf (stderr, "variable_new: end\n");
00063     #endif
00064 }
00065
00066 void
00067 variable_free (Variable * variable,
00068               unsigned int type)
00069 {
00070     #if DEBUG_VARIABLE
00071         fprintf (stderr, "variable_free: start\n");
00072     #endif
00073     if (type == INPUT_TYPE_XML)
00074         xmlFree (variable->name);
00075     else
00076         g_free (variable->name);
00077     #if DEBUG_VARIABLE
00078         fprintf (stderr, "variable_free: end\n");
00079     #endif
00080 }
00081
00082 void
00083 variable_error (Variable * variable,
00084               char *message)
00085 {
00086     char buffer[64];
00087     if (!variable->name)
00088         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00089     else
00090         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00091     error_message = g_strdup (buffer);
00092 }
00093
00094 int
00095 variable_open_xml (Variable * variable,
00096                  xmlNode * node,
00097                  unsigned int algorithm,
00098                  unsigned int nsteps)
00099 {
00100     int error_code;
00101
00102     #if DEBUG_VARIABLE
00103         fprintf (stderr, "variable_open_xml: start\n");
00104     #endif
00105     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00106     if (!variable->name)
00107     {

```

```

00134     variable_error (variable, _("no name"));
00135     goto exit_on_error;
00136 }
00137 if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00138 {
00139     variable->rangemin
00140     = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00141                          &error_code);
00142     if (error_code)
00143     {
00144         variable_error (variable, _("bad minimum"));
00145         goto exit_on_error;
00146     }
00147     variable->rangeminabs = xml_node_get_float_with_default
00148     (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00149      &error_code);
00150     if (error_code)
00151     {
00152         variable_error (variable, _("bad absolute minimum"));
00153         goto exit_on_error;
00154     }
00155     if (variable->rangemin < variable->rangeminabs)
00156     {
00157         variable_error (variable, _("minimum range not allowed"));
00158         goto exit_on_error;
00159     }
00160 }
00161 else
00162 {
00163     variable_error (variable, _("no minimum range"));
00164     goto exit_on_error;
00165 }
00166 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00167 {
00168     variable->rangemax
00169     = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00170                          &error_code);
00171     if (error_code)
00172     {
00173         variable_error (variable, _("bad maximum"));
00174         goto exit_on_error;
00175     }
00176     variable->rangemaxabs = xml_node_get_float_with_default
00177     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00178      &error_code);
00179     if (error_code)
00180     {
00181         variable_error (variable, _("bad absolute maximum"));
00182         goto exit_on_error;
00183     }
00184     if (variable->rangemax > variable->rangemaxabs)
00185     {
00186         variable_error (variable, _("maximum range not allowed"));
00187         goto exit_on_error;
00188     }
00189     if (variable->rangemax < variable->rangemin)
00190     {
00191         variable_error (variable, _("bad range"));
00192         goto exit_on_error;
00193     }
00194 }
00195 else
00196 {
00197     variable_error (variable, _("no maximum range"));
00198     goto exit_on_error;
00199 }
00200 variable->precision
00201 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
00202                                  DEFAULT_PRECISION, &error_code);
00203 if (error_code || variable->precision >= NPRECISIONS)
00204 {
00205     variable_error (variable, _("bad precision"));
00206     goto exit_on_error;
00207 }
00208 if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
00209 {
00210     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00211     {
00212         variable->nsweeps
00213         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00214                              &error_code);
00215         if (error_code || !variable->nsweeps)
00216         {
00217             variable_error (variable, _("bad sweeps"));
00218             goto exit_on_error;

```

```

00219     }
00220 }
00221 else
00222 {
00223     variable_error (variable, _("no sweeps number"));
00224     goto exit_on_error;
00225 }
00226 #if DEBUG_VARIABLE
00227     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00228 #endif
00229 }
00230 if (algorithm == ALGORITHM_GENETIC)
00231 {
00232     // Obtaining bits representing each variable
00233     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00234     {
00235         variable->nbits
00236         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00237                             &error_code);
00238         if (error_code || !variable->nbits)
00239         {
00240             variable_error (variable, _("invalid bits number"));
00241             goto exit_on_error;
00242         }
00243     }
00244     else
00245     {
00246         variable_error (variable, _("no bits number"));
00247         goto exit_on_error;
00248     }
00249 }
00250 else if (nsteps)
00251 {
00252     variable->step
00253     = xml_node_get_float (node, (const xmlChar *)
00254                          LABEL_STEP, &error_code);
00255     if (error_code || variable->step < 0.)
00256     {
00257         variable_error (variable, _("bad step size"));
00258         goto exit_on_error;
00259     }
00260 }
00261 #if DEBUG_VARIABLE
00262     fprintf (stderr, "variable_open_xml: end\n");
00263 #endif
00264 return 1;
00265 exit_on_error:
00266     variable_free (variable, INPUT_TYPE_XML);
00267 #if DEBUG_VARIABLE
00268     fprintf (stderr, "variable_open_xml: end\n");
00269 #endif
00270 return 0;
00271 }
00272
00273 int
00274 variable_open_json (Variable * variable,
00275                    JsonNode * node,
00276                    unsigned int algorithm,
00277                    unsigned int nsteps)
00278 {
00279     JsonObject *object;
00280     const char *label;
00281     int error_code;
00282     #if DEBUG_VARIABLE
00283         fprintf (stderr, "variable_open_json: start\n");
00284     #endif
00285     object = json_node_get_object (node);
00286     label = json_object_get_string_member (object, LABEL_NAME);
00287     if (!label)
00288     {
00289         variable_error (variable, _("no name"));
00290         goto exit_on_error;
00291     }
00292     variable->name = g_strdup (label);
00293     if (json_object_get_member (object, LABEL_MINIMUM))
00294     {
00295         variable->rangemin
00296         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00297         if (error_code)
00298         {
00299             variable_error (variable, _("bad minimum"));
00300             goto exit_on_error;
00301         }
00302     }
00303     variable->rangeminabs
00304     = json_object_get_float_with_default (object,
00305                                           LABEL_ABSOLUTE_MINIMUM,

```

```

00310                                     -G_MAXDOUBLE, &error_code);
00311     if (error_code)
00312     {
00313         variable_error (variable, _("bad absolute minimum"));
00314         goto exit_on_error;
00315     }
00316     if (variable->rangemin < variable->rangeminabs)
00317     {
00318         variable_error (variable, _("minimum range not allowed"));
00319         goto exit_on_error;
00320     }
00321 }
00322 else
00323 {
00324     variable_error (variable, _("no minimum range"));
00325     goto exit_on_error;
00326 }
00327 if (json_object_get_member (object, LABEL_MAXIMUM))
00328 {
00329     variable->rangemax
00330     = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00331     if (error_code)
00332     {
00333         variable_error (variable, _("bad maximum"));
00334         goto exit_on_error;
00335     }
00336     variable->rangemaxabs
00337     = json_object_get_float_with_default (object,
00338 LABEL_ABSOLUTE_MAXIMUM,
00339                                     G_MAXDOUBLE, &error_code);
00340     if (error_code)
00341     {
00342         variable_error (variable, _("bad absolute maximum"));
00343         goto exit_on_error;
00344     }
00345     if (variable->rangemax > variable->rangemaxabs)
00346     {
00347         variable_error (variable, _("maximum range not allowed"));
00348         goto exit_on_error;
00349     }
00350     if (variable->rangemax < variable->rangemin)
00351     {
00352         variable_error (variable, _("bad range"));
00353         goto exit_on_error;
00354     }
00355 }
00356 else
00357 {
00358     variable_error (variable, _("no maximum range"));
00359     goto exit_on_error;
00360 }
00361 variable->precision
00362 = json_object_get_uint_with_default (object,
00363 LABEL_PRECISION,
00364                                     DEFAULT_PRECISION, &error_code);
00365 if (error_code || variable->precision >= NPRECISIONS)
00366 {
00367     variable_error (variable, _("bad precision"));
00368     goto exit_on_error;
00369 }
00370 if (algorithm == ALGORITHM_SWEEP || algorithm ==
00371 ALGORITHM_ORTHOGONAL)
00372 {
00373     if (json_object_get_member (object, LABEL_NSWEEPS))
00374     {
00375         variable->nsweeps
00376         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00377         if (error_code || !variable->nsweeps)
00378         {
00379             variable_error (variable, _("bad sweeps"));
00380             goto exit_on_error;
00381         }
00382     }
00383 }
00384 else
00385 {
00386     variable_error (variable, _("no sweeps number"));
00387     goto exit_on_error;
00388 }
00389 #if DEBUG_VARIABLE
00390 fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00391 #endif
00392 if (algorithm == ALGORITHM_GENETIC)
00393 {
00394     // Obtaining bits representing each variable
00395     if (json_object_get_member (object, LABEL_NBITS))
00396     {

```



```

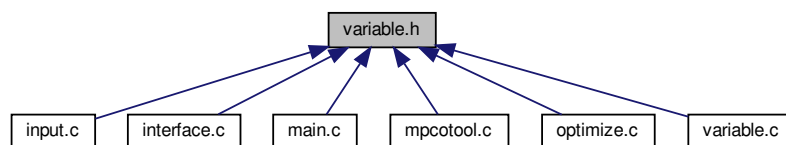
00394     variable->nbits
00395     = json_object_get_uint (object, LABEL_NBITS, &error_code);
00396     if (error_code || !variable->nbits)
00397     {
00398         variable_error (variable, _("invalid bits number"));
00399         goto exit_on_error;
00400     }
00401 }
00402 else
00403 {
00404     variable_error (variable, _("no bits number"));
00405     goto exit_on_error;
00406 }
00407 }
00408 else if (nsteps)
00409 {
00410     variable->step = json_object_get_float (object,
00411 LABEL_STEP, &error_code);
00412     if (error_code || variable->step < 0.)
00413     {
00414         variable_error (variable, _("bad step size"));
00415         goto exit_on_error;
00416     }
00417 }
00418 #if DEBUG_VARIABLE
00419 fprintf (stderr, "variable_open_json: end\n");
00420 #endif
00421 return 1;
00422 exit_on_error:
00423 variable_free (variable, INPUT_TYPE_JSON);
00424 #if DEBUG_VARIABLE
00425 fprintf (stderr, "variable_open_json: end\n");
00426 #endif
00427 return 0;
00428 }

```

4.29 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_new](#) ([Variable](#) *variable)
- void [variable_free](#) ([Variable](#) *variable, unsigned int type)
- void [variable_error](#) ([Variable](#) *variable, char *message)
- int [variable_open_xml](#) ([Variable](#) *variable, xmlDoc *node, unsigned int algorithm, unsigned int nsteps)
- int [variable_open_json](#) ([Variable](#) *variable, cJSON *node, unsigned int algorithm, unsigned int nsteps)

Variables

- const char * [format](#) [[NPRECISIONS](#)]
Array of C-strings with variable formats.
- const double [precision](#) [[NPRECISIONS](#)]
Array of variable precisions.

4.29.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.29.2 Enumeration Type Documentation

4.29.2.1 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 };
```

4.29.3 Function Documentation

4.29.3.1 [variable_error\(\)](#)

```
void variable_error (
    Variable * variable,
    char * message )
```

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

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	Error message.

Definition at line 100 of file [variable.c](#).

```
00104 {
00105     char buffer[64];
00106     if (!variable->name)
00107         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00108     else
00109         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00110     error_message = g_strdup (buffer);
00111 }
```

4.29.3.2 [variable_free\(\)](#)

```
void variable_free (
    Variable * variable,
    unsigned int type )
```

Function to free the memory of a [Variable](#) struct.

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	Type of input file.

Definition at line 79 of file [variable.c](#).

```
00083 {
00084     #if DEBUG_VARIABLE
00085     fprintf (stderr, "variable_free: start\n");
00086     #endif
00087     if (type == INPUT_TYPE_XML)
00088         xmlFree (variable->name);
00089     else
00090         g_free (variable->name);
00091     #if DEBUG_VARIABLE
00092     fprintf (stderr, "variable_free: end\n");
00093     #endif
00094 }
```

4.29.3.3 variable_new()

```
void variable_new (
    Variable * variable )
```

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

Parameters

<i>variable</i>	Variable struct.
-----------------	----------------------------------

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

```
00065 {
00066     #if DEBUG_VARIABLE
00067     fprintf (stderr, "variable_new: start\n");
00068     #endif
00069     variable->name = NULL;
00070     #if DEBUG_VARIABLE
00071     fprintf (stderr, "variable_new: end\n");
00072     #endif
00073 }
```

4.29.3.4 variable_open_json()

```
int variable_open_json (
    Variable * variable,
    JsonNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 279 of file [variable.c](#).

```

00284 {
00285     JsonObject *object;
00286     const char *label;
00287     int error_code;
00288     #if DEBUG_VARIABLE
00289     fprintf (stderr, "variable_open_json: start\n");
00290     #endif
00291     object = json_node_get_object (node);
00292     label = json_object_get_string_member (object, LABEL_NAME);
00293     if (!label)
00294     {
00295         variable_error (variable, _("no name"));
00296         goto exit_on_error;
00297     }
00298     variable->name = g_strdup (label);
00299     if (json_object_get_member (object, LABEL_MINIMUM))
00300     {
00301         variable->rangemin
00302         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00303         if (error_code)
00304         {
00305             variable_error (variable, _("bad minimum"));
00306             goto exit_on_error;
00307         }
00308         variable->rangeminabs
00309         = json_object_get_float_with_default (object,
00310 LABEL_ABSOLUTE_MINIMUM,
00311                                             -G_MAXDOUBLE, &error_code);
00312         if (error_code)
00313         {
00314             variable_error (variable, _("bad absolute minimum"));
00315             goto exit_on_error;
00316         }
00317         if (variable->rangemin < variable->rangeminabs)
00318         {
00319             variable_error (variable, _("minimum range not allowed"));
00320             goto exit_on_error;
00321         }
00322     }
00323     else
00324     {
00325         variable_error (variable, _("no minimum range"));
00326         goto exit_on_error;
00327     }
00328     if (json_object_get_member (object, LABEL_MAXIMUM))
00329     {
00330         variable->rangemax
00331         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00332         if (error_code)
00333         {
00334             variable_error (variable, _("bad maximum"));
00335             goto exit_on_error;
00336         }
00337         variable->rangemaxabs
00338         = json_object_get_float_with_default (object,
00339 LABEL_ABSOLUTE_MAXIMUM,
00340                                             G_MAXDOUBLE, &error_code);
00341         if (error_code)
00342         {
00343             variable_error (variable, _("bad absolute maximum"));
00344             goto exit_on_error;
00345         }
00346         if (variable->rangemax > variable->rangemaxabs)
00347         {
00348             variable_error (variable, _("maximum range not allowed"));
00349             goto exit_on_error;
00350         }
00351         if (variable->rangemax < variable->rangemin)
00352         {
00353             variable_error (variable, _("bad range"));
00354         }
00355     }

```

```

00352         goto exit_on_error;
00353     }
00354 }
00355 else
00356 {
00357     variable_error (variable, _("no maximum range"));
00358     goto exit_on_error;
00359 }
00360 variable->precision
00361 = json_object_get_uint_with_default (object,
LABEL_PRECISION,
00362                                     DEFAULT_PRECISION, &error_code);
00363 if (error_code || variable->precision >= NPRECISIONS)
00364 {
00365     variable_error (variable, _("bad precision"));
00366     goto exit_on_error;
00367 }
00368 if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
00369 {
00370     if (json_object_get_member (object, LABEL_NSWEEPS))
00371     {
00372         variable->nsweeps
00373         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00374         if (error_code || !variable->nsweeps)
00375         {
00376             variable_error (variable, _("bad sweeps"));
00377             goto exit_on_error;
00378         }
00379     }
00380     else
00381     {
00382         variable_error (variable, _("no sweeps number"));
00383         goto exit_on_error;
00384     }
00385 #if DEBUG_VARIABLE
00386     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00387 #endif
00388 }
00389 if (algorithm == ALGORITHM_GENETIC)
00390 {
00391     // Obtaining bits representing each variable
00392     if (json_object_get_member (object, LABEL_NBITS))
00393     {
00394         variable->nbits
00395         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00396         if (error_code || !variable->nbits)
00397         {
00398             variable_error (variable, _("invalid bits number"));
00399             goto exit_on_error;
00400         }
00401     }
00402     else
00403     {
00404         variable_error (variable, _("no bits number"));
00405         goto exit_on_error;
00406     }
00407 }
00408 else if (nsteps)
00409 {
00410     variable->step = json_object_get_float (object,
LABEL_STEP, &error_code);
00411     if (error_code || variable->step < 0.)
00412     {
00413         variable_error (variable, _("bad step size"));
00414         goto exit_on_error;
00415     }
00416 }
00417 #if DEBUG_VARIABLE
00418     fprintf (stderr, "variable_open_json: end\n");
00419 #endif
00420 return 1;
00421 exit_on_error:
00422     variable_free (variable, INPUT_TYPE_JSON);
00423 #if DEBUG_VARIABLE
00424     fprintf (stderr, "variable_open_json: end\n");
00425 #endif
00426 return 0;
00427 }
00428 }

```

Here is the call graph for this function:



4.29.3.5 variable_open_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
  
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 119 of file [variable.c](#).

```

00124 {
00125     int error_code;
00126
00127     #if DEBUG_VARIABLE
00128         fprintf (stderr, "variable_open_xml: start\n");
00129     #endif
00130
00131     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00132     if (!variable->name)
00133     {
00134         variable_error (variable, _("no name"));
00135         goto exit_on_error;
00136     }
00137     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00138     {
00139         variable->rangemin
00140             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00141                                   &error_code);
00142         if (error_code)
00143         {
00144             variable_error (variable, _("bad minimum"));
00145             goto exit_on_error;
00146         }
00147     }
00148 }
  
```

```

00146     }
00147     variable->rangeminabs = xml_node_get_float_with_default
00148     (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00149     &error_code);
00150     if (error_code)
00151     {
00152         variable_error (variable, _("bad absolute minimum"));
00153         goto exit_on_error;
00154     }
00155     if (variable->rangemin < variable->rangeminabs)
00156     {
00157         variable_error (variable, _("minimum range not allowed"));
00158         goto exit_on_error;
00159     }
00160 }
00161 else
00162 {
00163     variable_error (variable, _("no minimum range"));
00164     goto exit_on_error;
00165 }
00166 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00167 {
00168     variable->rangemax
00169     = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00170     &error_code);
00171     if (error_code)
00172     {
00173         variable_error (variable, _("bad maximum"));
00174         goto exit_on_error;
00175     }
00176     variable->rangemaxabs = xml_node_get_float_with_default
00177     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00178     &error_code);
00179     if (error_code)
00180     {
00181         variable_error (variable, _("bad absolute maximum"));
00182         goto exit_on_error;
00183     }
00184     if (variable->rangemax > variable->rangemaxabs)
00185     {
00186         variable_error (variable, _("maximum range not allowed"));
00187         goto exit_on_error;
00188     }
00189     if (variable->rangemax < variable->rangemin)
00190     {
00191         variable_error (variable, _("bad range"));
00192         goto exit_on_error;
00193     }
00194 }
00195 else
00196 {
00197     variable_error (variable, _("no maximum range"));
00198     goto exit_on_error;
00199 }
00200 variable->precision
00201 = xml_node_get_uint_with_default (node, (const xmlChar *)
00202 LABEL_PRECISION,
00203                                     DEFAULT_PRECISION, &error_code);
00204 if (error_code || variable->precision >= NPRECISIONS)
00205 {
00206     variable_error (variable, _("bad precision"));
00207     goto exit_on_error;
00208 }
00209 if (algorithm == ALGORITHM_SWEEP || algorithm ==
00210 ALGORITHM_ORTHOGONAL)
00211 {
00212     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00213     {
00214         variable->nsweeps
00215         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00216         &error_code);
00217         if (error_code || !variable->nsweeps)
00218         {
00219             variable_error (variable, _("bad sweeps"));
00220             goto exit_on_error;
00221         }
00222     }
00223     else
00224     {
00225         variable_error (variable, _("no sweeps number"));
00226         goto exit_on_error;
00227     }
00228 }
00229 #if DEBUG_VARIABLE
00230 fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00231 #endif
00232 if (algorithm == ALGORITHM_GENETIC)

```



```

00231     {
00232         // Obtaining bits representing each variable
00233         if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00234         {
00235             variable->nbits
00236             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00237                                 &error_code);
00238             if (error_code || !variable->nbits)
00239             {
00240                 variable_error (variable, _("invalid bits number"));
00241                 goto exit_on_error;
00242             }
00243         }
00244         else
00245         {
00246             variable_error (variable, _("no bits number"));
00247             goto exit_on_error;
00248         }
00249     }
00250     else if (nsteps)
00251     {
00252         variable->step
00253         = xml_node_get_float (node, (const xmlChar *)
00254                               LABEL_STEP, &error_code);
00255         if (error_code || variable->step < 0.)
00256         {
00257             variable_error (variable, _("bad step size"));
00258             goto exit_on_error;
00259         }
00260     }
00261     #if DEBUG_VARIABLE
00262     fprintf (stderr, "variable_open_xml: end\n");
00263     #endif
00264     return 1;
00265 exit_on_error:
00266     variable_free (variable, INPUT_TYPE_XML);
00267     #if DEBUG_VARIABLE
00268     fprintf (stderr, "variable_open_xml: end\n");
00269     #endif
00270     return 0;
00271 }

```

Here is the call graph for this function:



4.30 variable.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013 1. Redistributions of source code must retain the above copyright notice,
00014    this list of conditions and the following disclaimer.
00015
00016 2. Redistributions in binary form must reproduce the above copyright notice,
00017    this list of conditions and the following disclaimer in the

```

```
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef VARIABLE__H
00033 #define VARIABLE__H 1
00034
00035 enum Algorithm
00036 {
00037     ALGORITHM_MONTE_CARLO = 0,
00038     ALGORITHM_SWEEP = 1,
00039     ALGORITHM_GENETIC = 2,
00040     ALGORITHM_ORTHOGONAL = 3
00041 };
00042
00043 typedef struct
00044 {
00045     char *name;
00046     double rangemin;
00047     double rangemax;
00048     double rangeminabs;
00049     double rangemaxabs;
00050     double step;
00051     unsigned int precision;
00052     unsigned int nsweeps;
00053     unsigned int nbits;
00054 } Variable;
00055
00056 extern const char *format[NPRECISIONS];
00057 extern const double precision[NPRECISIONS];
00058
00059 // Public functions
00060 void variable_new (Variable * variable);
00061 void variable_free (Variable * variable, unsigned int type);
00062 void variable_error (Variable * variable, char *message);
00063 int variable_open_xml (Variable * variable, xmlNode * node,
00064                       unsigned int algorithm, unsigned int nsteps);
00065 int variable_open_json (Variable * variable, JsonNode * node,
00066                        unsigned int algorithm, unsigned int nsteps);
00067
00068 #endif
```

Index

Algorithm
variable.h, 348

ClimbingMethod
input.h, 70
config.h, 19
INPUT_TYPE, 22
cores_number
utils.c, 301
utils.h, 320

ErrorNorm
input.h, 71
Experiment, 5
experiment.c, 24
experiment_error, 25
experiment_free, 26
experiment_new, 26
experiment_open_json, 27
experiment_open_xml, 29
stencil, 31
experiment.h, 35
experiment_error, 36
experiment_free, 36
experiment_new, 37
experiment_open_json, 37
experiment_open_xml, 39
experiment_error
experiment.c, 25
experiment.h, 36
experiment_free
experiment.c, 26
experiment.h, 36
experiment_new
experiment.c, 26
experiment.h, 37
experiment_open_json
experiment.c, 27
experiment.h, 37
experiment_open_xml
experiment.c, 29
experiment.h, 39

format
variable.c, 342

gtk_array_get_active
interface.h, 167
utils.c, 301
utils.h, 320

INPUT_TYPE
config.h, 22
Input, 6
input.c, 42
input_error, 43
input_free, 44
input_new, 44
input_open, 45
input_open_json, 46
input_open_xml, 51
input.h, 69
ClimbingMethod, 70
ErrorNorm, 71
input_error, 71
input_free, 72
input_new, 72
input_open, 73
input_open_json, 74
input_open_xml, 79
input_error
input.c, 43
input.h, 71
input_free
input.c, 44
input.h, 72
input_new
input.c, 44
input.h, 72
input_open
input.c, 45
input.h, 73
input_open_json
input.c, 46
input.h, 74
input_open_xml
input.c, 51
input.h, 79
input_save
interface.c, 88
interface.h, 168
input_save_climbing_json
interface.c, 89
input_save_climbing_xml
interface.c, 90
input_save_json
interface.c, 92
input_save_xml
interface.c, 95
interface.c, 86

- input_save, 88
- input_save_climbing_json, 89
- input_save_climbing_xml, 90
- input_save_json, 92
- input_save_xml, 95
- options_new, 98
- running_new, 99
- window_about, 99
- window_add_experiment, 100
- window_add_variable, 101
- window_get_algorithm, 102
- window_get_climbing, 102
- window_get_norm, 103
- window_help, 104
- window_inputs_experiment, 104
- window_label_variable, 105
- window_name_experiment, 105
- window_new, 106
- window_open, 115
- window_precision_variable, 116
- window_rangemax_variable, 116
- window_rangemaxabs_variable, 117
- window_rangemin_variable, 117
- window_rangeminabs_variable, 117
- window_read, 118
- window_remove_experiment, 120
- window_remove_variable, 121
- window_run, 122
- window_save, 123
- window_save_climbing, 125
- window_set_algorithm, 126
- window_set_experiment, 126
- window_set_variable, 127
- window_step_variable, 128
- window_template_experiment, 129
- window_update, 129
- window_update_climbing, 132
- window_update_variable, 132
- window_weight_experiment, 133
- interface.h, 165
 - gtk_array_get_active, 167
 - input_save, 168
 - options_new, 169
 - running_new, 170
 - window_add_experiment, 170
 - window_add_variable, 172
 - window_get_algorithm, 172
 - window_get_climbing, 173
 - window_get_norm, 174
 - window_help, 174
 - window_inputs_experiment, 175
 - window_label_variable, 175
 - window_name_experiment, 176
 - window_new, 176
 - window_open, 185
 - window_precision_variable, 186
 - window_rangemax_variable, 187
 - window_rangemaxabs_variable, 187
 - window_rangemin_variable, 187
 - window_rangeminabs_variable, 188
 - window_read, 188
 - window_remove_experiment, 190
 - window_remove_variable, 191
 - window_run, 192
 - window_save, 193
 - window_save_climbing, 195
 - window_set_algorithm, 196
 - window_set_experiment, 197
 - window_set_variable, 197
 - window_template_experiment, 199
 - window_update, 199
 - window_update_climbing, 202
 - window_update_variable, 202
 - window_weight_experiment, 203
- json_object_get_float
 - utils.c, 302
 - utils.h, 321
- json_object_get_float_with_default
 - utils.c, 302
 - utils.h, 321
- json_object_get_int
 - utils.c, 303
 - utils.h, 322
- json_object_get_uint
 - utils.c, 304
 - utils.h, 323
- json_object_get_uint_with_default
 - utils.c, 305
 - utils.h, 323
- json_object_set_float
 - utils.c, 306
 - utils.h, 324
- json_object_set_int
 - utils.c, 306
 - utils.h, 325
- json_object_set_uint
 - utils.c, 306
 - utils.h, 325
- main.c, 206
- mpcotool
 - mpcotool.c, 209
- mpcotool.c, 208
- mpcotool, 209
- Optimize, 7
 - thread_climbing, 10
- optimize.c, 214
 - optimize_MonteCarlo, 228
 - optimize_best, 216
 - optimize_best_climbing, 217
 - optimize_climbing, 218
 - optimize_climbing_sequential, 219
 - optimize_climbing_thread, 220
 - optimize_estimate_climbing_coordinates, 221
 - optimize_estimate_climbing_random, 221

- optimize_free, 222
- optimize_genetic, 222
- optimize_genetic_objective, 223
- optimize_input, 224
- optimize_iterate, 225
- optimize_merge, 226
- optimize_merge_old, 227
- optimize_norm_euclidian, 229
- optimize_norm_maximum, 229
- optimize_norm_p, 230
- optimize_norm_taxicab, 231
- optimize_open, 232
- optimize_orthogonal, 236
- optimize_parse, 237
- optimize_print, 239
- optimize_refine, 239
- optimize_save_old, 241
- optimize_save_variables, 241
- optimize_sequential, 242
- optimize_step, 243
- optimize_step_climbing, 243
- optimize_sweep, 244
- optimize_synchronise, 245
- optimize_thread, 246
- optimize.h, 265
 - optimize_MonteCarlo, 279
 - optimize_best, 267
 - optimize_best_climbing, 268
 - optimize_climbing, 268
 - optimize_climbing_sequential, 269
 - optimize_climbing_thread, 270
 - optimize_estimate_climbing_coordinates, 271
 - optimize_estimate_climbing_random, 272
 - optimize_free, 272
 - optimize_genetic, 273
 - optimize_genetic_objective, 274
 - optimize_input, 275
 - optimize_iterate, 276
 - optimize_merge, 277
 - optimize_merge_old, 278
 - optimize_norm_euclidian, 279
 - optimize_norm_maximum, 280
 - optimize_norm_p, 281
 - optimize_norm_taxicab, 282
 - optimize_open, 283
 - optimize_orthogonal, 287
 - optimize_parse, 288
 - optimize_print, 290
 - optimize_refine, 290
 - optimize_save_old, 292
 - optimize_save_variables, 292
 - optimize_sequential, 293
 - optimize_step, 294
 - optimize_step_climbing, 294
 - optimize_sweep, 295
 - optimize_synchronise, 296
 - optimize_thread, 297
- optimize_MonteCarlo
 - optimize.c, 228
 - optimize.h, 279
- optimize_best
 - optimize.c, 216
 - optimize.h, 267
- optimize_best_climbing
 - optimize.c, 217
 - optimize.h, 268
- optimize_climbing
 - optimize.c, 218
 - optimize.h, 268
- optimize_climbing_sequential
 - optimize.c, 219
 - optimize.h, 269
- optimize_climbing_thread
 - optimize.c, 220
 - optimize.h, 270
- optimize_estimate_climbing_coordinates
 - optimize.c, 221
 - optimize.h, 271
- optimize_estimate_climbing_random
 - optimize.c, 221
 - optimize.h, 272
- optimize_free
 - optimize.c, 222
 - optimize.h, 272
- optimize_genetic
 - optimize.c, 222
 - optimize.h, 273
- optimize_genetic_objective
 - optimize.c, 223
 - optimize.h, 274
- optimize_input
 - optimize.c, 224
 - optimize.h, 275
- optimize_iterate
 - optimize.c, 225
 - optimize.h, 276
- optimize_merge
 - optimize.c, 226
 - optimize.h, 277
- optimize_merge_old
 - optimize.c, 227
 - optimize.h, 278
- optimize_norm_euclidian
 - optimize.c, 229
 - optimize.h, 279
- optimize_norm_maximum
 - optimize.c, 229
 - optimize.h, 280
- optimize_norm_p
 - optimize.c, 230
 - optimize.h, 281
- optimize_norm_taxicab
 - optimize.c, 231
 - optimize.h, 282
- optimize_open
 - optimize.c, 232

- optimize.h, 283
- optimize_orthogonal
 - optimize.c, 236
 - optimize.h, 287
- optimize_parse
 - optimize.c, 237
 - optimize.h, 288
- optimize_print
 - optimize.c, 239
 - optimize.h, 290
- optimize_refine
 - optimize.c, 239
 - optimize.h, 290
- optimize_save_old
 - optimize.c, 241
 - optimize.h, 292
- optimize_save_variables
 - optimize.c, 241
 - optimize.h, 292
- optimize_sequential
 - optimize.c, 242
 - optimize.h, 293
- optimize_step
 - optimize.c, 243
 - optimize.h, 294
- optimize_step_climbing
 - optimize.c, 243
 - optimize.h, 294
- optimize_sweep
 - optimize.c, 244
 - optimize.h, 295
- optimize_synchronise
 - optimize.c, 245
 - optimize.h, 296
- optimize_thread
 - optimize.c, 246
 - optimize.h, 297
- Options, 11
- options_new
 - interface.c, 98
 - interface.h, 169
- ParallelData, 11
- precision
 - variable.c, 342
- process_pending
 - utils.c, 307
 - utils.h, 326
- Running, 12
- running_new
 - interface.c, 99
 - interface.h, 170
- show_error
 - utils.c, 307
 - utils.h, 326
- show_message
 - utils.c, 308
- utils.h, 327
- stencil
 - experiment.c, 31
- thread_climbing
 - Optimize, 10
- utils.c, 300
 - cores_number, 301
 - gtk_array_get_active, 301
 - json_object_get_float, 302
 - json_object_get_float_with_default, 302
 - json_object_get_int, 303
 - json_object_get_uint, 304
 - json_object_get_uint_with_default, 305
 - json_object_set_float, 306
 - json_object_set_int, 306
 - json_object_set_uint, 306
 - process_pending, 307
 - show_error, 307
 - show_message, 308
 - xml_node_get_float, 308
 - xml_node_get_float_with_default, 309
 - xml_node_get_int, 310
 - xml_node_get_uint, 311
 - xml_node_get_uint_with_default, 311
 - xml_node_set_float, 312
 - xml_node_set_int, 313
 - xml_node_set_uint, 313
- utils.h, 318
 - cores_number, 320
 - gtk_array_get_active, 320
 - json_object_get_float, 321
 - json_object_get_float_with_default, 321
 - json_object_get_int, 322
 - json_object_get_uint, 323
 - json_object_get_uint_with_default, 323
 - json_object_set_float, 324
 - json_object_set_int, 325
 - json_object_set_uint, 325
 - process_pending, 326
 - show_error, 326
 - show_message, 327
 - xml_node_get_float, 327
 - xml_node_get_float_with_default, 328
 - xml_node_get_int, 329
 - xml_node_get_uint, 330
 - xml_node_get_uint_with_default, 330
 - xml_node_set_float, 331
 - xml_node_set_int, 332
 - xml_node_set_uint, 332
- Variable, 12
- variable.c, 334
 - format, 342
 - precision, 342
 - variable_error, 335
 - variable_free, 336
 - variable_new, 336

- variable_open_json, [337](#)
 - variable_open_xml, [339](#)
- variable.h, [347](#)
 - Algorithm, [348](#)
 - variable_error, [349](#)
 - variable_free, [349](#)
 - variable_new, [350](#)
 - variable_open_json, [350](#)
 - variable_open_xml, [353](#)
- variable_error
 - variable.c, [335](#)
 - variable.h, [349](#)
- variable_free
 - variable.c, [336](#)
 - variable.h, [349](#)
- variable_new
 - variable.c, [336](#)
 - variable.h, [350](#)
- variable_open_json
 - variable.c, [337](#)
 - variable.h, [350](#)
- variable_open_xml
 - variable.c, [339](#)
 - variable.h, [353](#)
- Window, [13](#)
- window_about
 - interface.c, [99](#)
- window_add_experiment
 - interface.c, [100](#)
 - interface.h, [170](#)
- window_add_variable
 - interface.c, [101](#)
 - interface.h, [172](#)
- window_get_algorithm
 - interface.c, [102](#)
 - interface.h, [172](#)
- window_get_climbing
 - interface.c, [102](#)
 - interface.h, [173](#)
- window_get_norm
 - interface.c, [103](#)
 - interface.h, [174](#)
- window_help
 - interface.c, [104](#)
 - interface.h, [174](#)
- window_inputs_experiment
 - interface.c, [104](#)
 - interface.h, [175](#)
- window_label_variable
 - interface.c, [105](#)
 - interface.h, [175](#)
- window_name_experiment
 - interface.c, [105](#)
 - interface.h, [176](#)
- window_new
 - interface.c, [106](#)
 - interface.h, [176](#)
- window_open
 - interface.c, [115](#)
 - interface.h, [185](#)
- window_precision_variable
 - interface.c, [116](#)
 - interface.h, [186](#)
- window_rangemax_variable
 - interface.c, [116](#)
 - interface.h, [187](#)
- window_rangemaxabs_variable
 - interface.c, [117](#)
 - interface.h, [187](#)
- window_rangemin_variable
 - interface.c, [117](#)
 - interface.h, [187](#)
- window_rangeminabs_variable
 - interface.c, [117](#)
 - interface.h, [188](#)
- window_read
 - interface.c, [118](#)
 - interface.h, [188](#)
- window_remove_experiment
 - interface.c, [120](#)
 - interface.h, [190](#)
- window_remove_variable
 - interface.c, [121](#)
 - interface.h, [191](#)
- window_run
 - interface.c, [122](#)
 - interface.h, [192](#)
- window_save
 - interface.c, [123](#)
 - interface.h, [193](#)
- window_save_climbing
 - interface.c, [125](#)
 - interface.h, [195](#)
- window_set_algorithm
 - interface.c, [126](#)
 - interface.h, [196](#)
- window_set_experiment
 - interface.c, [126](#)
 - interface.h, [197](#)
- window_set_variable
 - interface.c, [127](#)
 - interface.h, [197](#)
- window_step_variable
 - interface.c, [128](#)
- window_template_experiment
 - interface.c, [129](#)
 - interface.h, [199](#)
- window_update
 - interface.c, [129](#)
 - interface.h, [199](#)
- window_update_climbing
 - interface.c, [132](#)
 - interface.h, [202](#)
- window_update_variable
 - interface.c, [132](#)
 - interface.h, [202](#)

window_weight_experiment
 interface.c, [133](#)
 interface.h, [203](#)

xml_node_get_float
 utils.c, [308](#)
 utils.h, [327](#)

xml_node_get_float_with_default
 utils.c, [309](#)
 utils.h, [328](#)

xml_node_get_int
 utils.c, [310](#)
 utils.h, [329](#)

xml_node_get_uint
 utils.c, [311](#)
 utils.h, [330](#)

xml_node_get_uint_with_default
 utils.c, [311](#)
 utils.h, [330](#)

xml_node_set_float
 utils.c, [312](#)
 utils.h, [331](#)

xml_node_set_int
 utils.c, [313](#)
 utils.h, [332](#)

xml_node_set_uint
 utils.c, [313](#)
 utils.h, [332](#)