

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_open_json()	37
4.5.2.4 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()	43

4.7.2.3	input_new()	44
4.7.2.4	input_open()	45
4.7.2.5	input_open_json()	46
4.7.2.6	input_open_xml()	51
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_free()	71
4.9.3.2	input_new()	72
4.9.3.3	input_open()	72
4.10	input.h	74
4.11	interface.c File Reference	75
4.11.1	Detailed Description	76
4.11.2	Function Documentation	77
4.11.2.1	input_save()	77
4.11.2.2	input_save_climbing_json()	78
4.11.2.3	input_save_climbing_xml()	79
4.11.2.4	input_save_json()	80
4.11.2.5	input_save_xml()	83
4.11.2.6	options_new()	85
4.11.2.7	running_new()	86
4.11.2.8	window_about()	87
4.11.2.9	window_add_experiment()	88
4.11.2.10	window_add_variable()	89
4.11.2.11	window_get_algorithm()	90
4.11.2.12	window_get_climbing()	90

4.11.2.13 window_get_norm()	91
4.11.2.14 window_help()	92
4.11.2.15 window_inputs_experiment()	92
4.11.2.16 window_label_variable()	93
4.11.2.17 window_name_experiment()	93
4.11.2.18 window_new()	93
4.11.2.19 window_open()	102
4.11.2.20 window_precision_variable()	103
4.11.2.21 window_rangemax_variable()	104
4.11.2.22 window_rangemaxabs_variable()	104
4.11.2.23 window_rangemin_variable()	105
4.11.2.24 window_rangeminabs_variable()	105
4.11.2.25 window_read()	105
4.11.2.26 window_remove_experiment()	107
4.11.2.27 window_remove_variable()	108
4.11.2.28 window_run()	109
4.11.2.29 window_save()	110
4.11.2.30 window_save_climbing()	112
4.11.2.31 window_set_algorithm()	113
4.11.2.32 window_set_experiment()	114
4.11.2.33 window_set_variable()	115
4.11.2.34 window_step_variable()	116
4.11.2.35 window_template_experiment()	116
4.11.2.36 window_update()	117
4.11.2.37 window_update_climbing()	119
4.11.2.38 window_update_variable()	120
4.11.2.39 window_weight_experiment()	121
4.12 interface.c	121
4.13 interface.h File Reference	153
4.13.1 Detailed Description	154

4.13.2	Function Documentation	154
4.13.2.1	window_new()	154
4.14	interface.h	163
4.15	main.c File Reference	165
4.15.1	Detailed Description	166
4.16	main.c	166
4.17	mpcotool.c File Reference	167
4.17.1	Detailed Description	168
4.17.2	Function Documentation	168
4.17.2.1	mpcotool()	168
4.18	mpcotool.c	171
4.19	mpcotool.h File Reference	174
4.19.1	Detailed Description	174
4.19.2	Function Documentation	175
4.19.2.1	mpcotool()	175
4.20	mpcotool.h	177
4.21	optimize.c File Reference	178
4.21.1	Detailed Description	180
4.21.2	Function Documentation	180
4.21.2.1	optimize_best()	180
4.21.2.2	optimize_best_climbing()	181
4.21.2.3	optimize_climbing()	181
4.21.2.4	optimize_climbing_sequential()	182
4.21.2.5	optimize_climbing_thread()	183
4.21.2.6	optimize_estimate_climbing_coordinates()	184
4.21.2.7	optimize_estimate_climbing_random()	185
4.21.2.8	optimize_free()	186
4.21.2.9	optimize_genetic()	186
4.21.2.10	optimize_genetic_objective()	187
4.21.2.11	optimize_input()	188

4.21.2.12 optimize_iterate()	189
4.21.2.13 optimize_merge()	190
4.21.2.14 optimize_merge_old()	191
4.21.2.15 optimize_MonteCarlo()	192
4.21.2.16 optimize_norm_euclidian()	193
4.21.2.17 optimize_norm_maximum()	194
4.21.2.18 optimize_norm_p()	195
4.21.2.19 optimize_norm_taxicab()	195
4.21.2.20 optimize_open()	196
4.21.2.21 optimize_orthogonal()	201
4.21.2.22 optimize_parse()	202
4.21.2.23 optimize_print()	203
4.21.2.24 optimize_refine()	204
4.21.2.25 optimize_save_old()	205
4.21.2.26 optimize_save_variables()	206
4.21.2.27 optimize_sequential()	206
4.21.2.28 optimize_step()	207
4.21.2.29 optimize_step_climbing()	207
4.21.2.30 optimize_sweep()	209
4.21.2.31 optimize_synchronise()	210
4.21.2.32 optimize_thread()	211
4.22 optimize.c	212
4.23 optimize.h File Reference	230
4.23.1 Detailed Description	231
4.23.2 Function Documentation	232
4.23.2.1 optimize_free()	232
4.23.2.2 optimize_open()	232
4.24 optimize.h	236
4.25 utils.c File Reference	238
4.25.1 Detailed Description	239

4.25.2	Function Documentation	239
4.25.2.1	cores_number()	240
4.25.2.2	gtk_array_get_active()	240
4.25.2.3	json_object_get_float()	241
4.25.2.4	json_object_get_float_with_default()	241
4.25.2.5	json_object_get_int()	242
4.25.2.6	json_object_get_uint()	243
4.25.2.7	json_object_get_uint_with_default()	243
4.25.2.8	json_object_set_float()	244
4.25.2.9	json_object_set_int()	245
4.25.2.10	json_object_set_uint()	245
4.25.2.11	process_pending()	246
4.25.2.12	show_error()	246
4.25.2.13	show_message()	247
4.25.2.14	xml_node_get_float()	247
4.25.2.15	xml_node_get_float_with_default()	248
4.25.2.16	xml_node_get_int()	249
4.25.2.17	xml_node_get_uint()	250
4.25.2.18	xml_node_get_uint_with_default()	250
4.25.2.19	xml_node_set_float()	251
4.25.2.20	xml_node_set_int()	252
4.25.2.21	xml_node_set_uint()	252
4.26	utils.c	253
4.27	utils.h File Reference	257
4.27.1	Detailed Description	258
4.27.2	Function Documentation	259
4.27.2.1	cores_number()	259
4.27.2.2	gtk_array_get_active()	259
4.27.2.3	json_object_get_float()	260
4.27.2.4	json_object_get_float_with_default()	260

4.27.2.5	<code>json_object_get_int()</code>	261
4.27.2.6	<code>json_object_get_uint()</code>	262
4.27.2.7	<code>json_object_get_uint_with_default()</code>	263
4.27.2.8	<code>json_object_set_float()</code>	263
4.27.2.9	<code>json_object_set_int()</code>	264
4.27.2.10	<code>json_object_set_uint()</code>	264
4.27.2.11	<code>process_pending()</code>	265
4.27.2.12	<code>show_error()</code>	265
4.27.2.13	<code>show_message()</code>	266
4.27.2.14	<code>xml_node_get_float()</code>	267
4.27.2.15	<code>xml_node_get_float_with_default()</code>	267
4.27.2.16	<code>xml_node_get_int()</code>	268
4.27.2.17	<code>xml_node_get_uint()</code>	269
4.27.2.18	<code>xml_node_get_uint_with_default()</code>	270
4.27.2.19	<code>xml_node_set_float()</code>	270
4.27.2.20	<code>xml_node_set_int()</code>	271
4.27.2.21	<code>xml_node_set_uint()</code>	271
4.28	<code>utils.h</code>	272
4.29	<code>variable.c</code> File Reference	273
4.29.1	Detailed Description	274
4.29.2	Function Documentation	274
4.29.2.1	<code>variable_error()</code>	274
4.29.2.2	<code>variable_free()</code>	275
4.29.2.3	<code>variable_open_json()</code>	275
4.29.2.4	<code>variable_open_xml()</code>	278
4.29.3	Variable Documentation	280
4.29.3.1	<code>format</code>	280
4.29.3.2	<code>precision</code>	280
4.30	<code>variable.c</code>	281
4.31	<code>variable.h</code> File Reference	285
4.31.1	Detailed Description	286
4.31.2	Enumeration Type Documentation	286
4.31.2.1	<code>Algorithm</code>	286
4.31.3	Function Documentation	287
4.31.3.1	<code>variable_error()</code>	287
4.31.3.2	<code>variable_free()</code>	287
4.31.3.3	<code>variable_open_json()</code>	288
4.31.3.4	<code>variable_open_xml()</code>	290
4.32	<code>variable.h</code>	293

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	75
interface.h	Header file to define the graphical interface functions	153
main.c	Main source file	165
mpcotool.c	Main function source file	167
mpcotool.h	Main function header file	174
optimize.c	Source file to define the optimization functions	178
optimize.h	Header file to define the optimization functions	230
utils.c	Source file to define some useful functions	238
utils.h	Header file to define some useful functions	257
variable.c	Source file to define the variable data	273
variable.h	Header file to define the variable data	285

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:

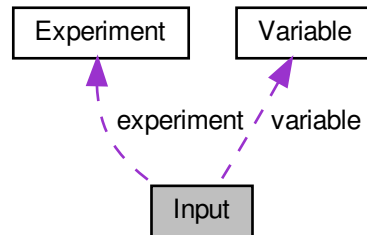
- [experiment.h](#)

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:

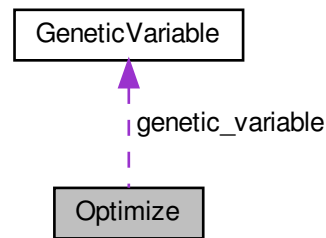
- [input.h](#)

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:

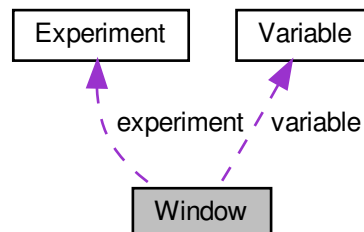
- [variable.h](#)

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.*
- GtkScrolledWindow * [scrolled_maxabs](#)
 - Absolute maximum GtkScrolledWindow.*
- GtkLabel * [label_precision](#)
 - Precision GtkLabel.*
- GtkSpinButton * [spin_precision](#)
 - Precision digits GtkSpinButton.*
- GtkLabel * [label_sweeps](#)
 - Sweeps number GtkLabel.*
- GtkSpinButton * [spin_sweeps](#)
 - Sweeps number GtkSpinButton.*
- GtkLabel * [label_bits](#)
 - Bits number GtkLabel.*
- GtkSpinButton * [spin_bits](#)
 - Bits number GtkSpinButton.*
- GtkLabel * [label_step](#)
 - GtkLabel to set the step.*
- GtkSpinButton * [spin_step](#)
 - GtkSpinButton to set the step.*
- GtkScrolledWindow * [scrolled_step](#)
 - step GtkScrolledWindow.*
- GtkFrame * [frame_experiment](#)
 - Experiment GtkFrame.*
- GtkGrid * [grid_experiment](#)
 - Experiment GtkGrid.*
- GtkComboBoxText * [combo_experiment](#)
 - Experiment GtkComboBoxEntry.*
- GtkButton * [button_add_experiment](#)
 - GtkButton to add a experiment.*
- GtkButton * [button_remove_experiment](#)
 - GtkButton to remove a experiment.*
- GtkLabel * [label_experiment](#)
 - Experiment GtkLabel.*
- GtkFileChooserButton * [button_experiment](#)
 - GtkFileChooserButton to set the experimental data file.*
- GtkLabel * [label_weight](#)
 - Weight GtkLabel.*
- GtkSpinButton * [spin_weight](#)
 - Weight GtkSpinButton.*
- GtkCheckButton * [check_template](#) [MAX_NINPUTS]
 - Array of GtkCheckButtons to set the input templates.*
- GtkFileChooserButton * [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](#)

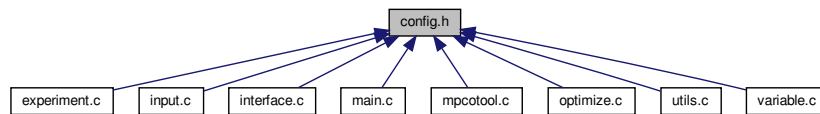
Chapter 4

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

INPUT_TYPE_XML	XML input file.
INPUT_TYPE_JSON	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
00033 #ifndef CONFIG__H
00034 #define CONFIG__H 1
00035
00036 // Gettext simplification
00037 #define _(string) (gettext(string))
00038
00039 // Array sizes
00040
00041 #define MAX_NINPUTS 8
00042 #define NALGORITHMS 4
00043 #define NCLIMBINGS 2
00044 #define NNORMS 4
00045 #define NPRECISIONS 15
00046
00047 // Default choices
00048 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00049 #define DEFAULT_RANDOM_SEED 7007
00050 #define DEFAULT_RELAXATION 1.
00051
00052 // Interface labels
00053
00054 #define LOCALE_DIR "locales"
00055 #define PROGRAM_INTERFACE "mpcotool"
00056
00057 // Labels
00058 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00059 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00060 #define LABEL_ADAPTATION "adaptation"
00061 #define LABEL_ALGORITHM "algorithm"
00062 #define LABEL_CLIMBING "climbing"
00063 #define LABEL_COORDINATES "coordinates"
00064 #define LABEL_EUCLIDIAN "euclidian"
00065 #define LABEL_EVALUATOR "evaluator"
00066 #define LABEL_EXPERIMENT "experiment"
00067 #define LABEL_EXPERIMENTS "experiments"
00068 #define LABEL_GENETIC "genetic"
00069 #define LABEL_MINIMUM "minimum"
00070 #define LABEL_MAXIMUM "maximum"
00071 #define LABEL_MONTE_CARLO "Monte-Carlo"
00072 #define LABEL_MUTATION "mutation"
00073 #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

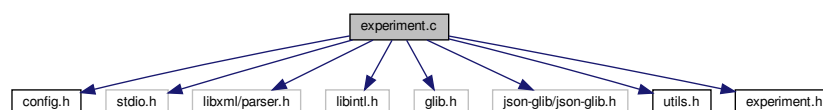
Source file to define the experiment data.

```

#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

- static void [experiment_new](#) ([Experiment](#) *experiment)
- void [experiment_free](#) ([Experiment](#) *experiment, unsigned int type)
- void [experiment_error](#) ([Experiment](#) *experiment, char *message)
- int [experiment_open_xml](#) ([Experiment](#) *experiment, xmlNode *node, unsigned int ninputs)
- int [experiment_open_json](#) ([Experiment](#) *experiment, JsonNode *node, unsigned int ninputs)

Variables

- const char * [stencil](#) [[MAX_NINPUTS](#)]
Array of xmlChar strings with stencil labels.

4.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

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()

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

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:

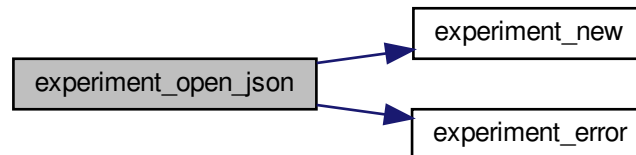
```

```

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);

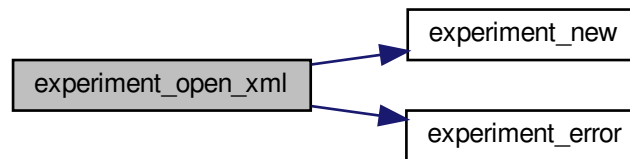
```

```

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",
00171                 experiment->name, stencil[0]);
00172     #endif
00173     ++experiment->ninputs;
00174 }
00175 else
00176 {
00177     experiment_error (experiment, _("no template"));
00178     goto exit_on_error;
00179 }
00180 for (i = 1; i < MAX_NINPUTS; ++i)
00181 {
00182     #if DEBUG_EXPERIMENT
00183         fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00184     #endif
00185     if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00186     {
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:



4.3.3 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 static 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

```

```

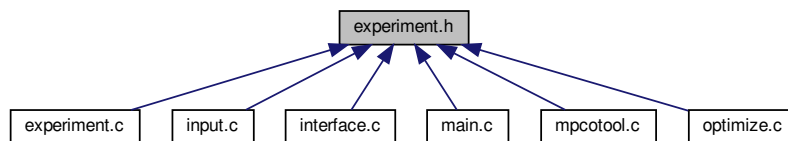
00307         experiment->stencil[i] = g_strdup (name);
00308         ++experiment->ninputs;
00309     }
00310     else if (ninputs && ninputs > i)
00311     {
00312         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00313         experiment_error (experiment, buffer);
00314         goto 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_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_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     #if DEBUG_EXPERIMENT
00278         fprintf (stderr, "experiment_open_json: experiment=%s template1=%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);

```

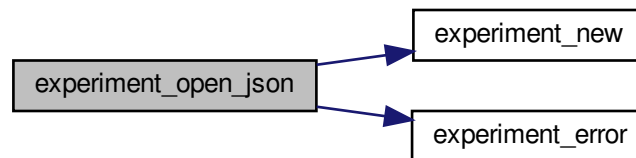


```

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

```

Here is the call graph for this function:



4.5.2.4 experiment_open_xml()

```

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

```

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

Returns

1 on success, 0 on error.

Parameters

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

Definition at line [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

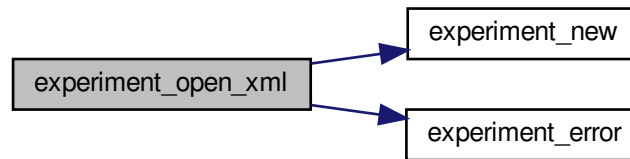
```

```

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     {
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->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     {
00211         break;
00212     }
00213 }
00214 #if DEBUG_EXPERIMENT
00215 fprintf (stderr, "experiment_open_xml: end\n");
00216 #endif
00217 return 1;
00218 exit_on_error:
00219 experiment_free (experiment, INPUT_TYPE_XML);
00220 #if DEBUG_EXPERIMENT
00221 fprintf (stderr, "experiment_open_xml: end\n");
00222 #endif
00223 return 0;
00224 }

```

Here is the call graph for this function:



4.6 experiment.h

```

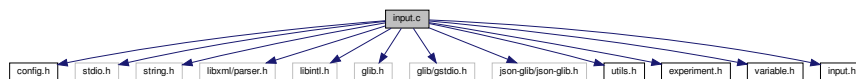
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use 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 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_free (Experiment * experiment, unsigned int type);
00047 void experiment_error (Experiment * experiment, char *message);
00048 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00049                        unsigned int ninputs);
00050 int experiment_open_json (Experiment * experiment, JsonNode * node,
00051                          unsigned int ninputs);
00052
00053 #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](#) ()
- static void [input_error](#) (char *message)
- static int [input_open_xml](#) (xmlDoc *doc)
- static int [input_open_json](#) (JsonParser *parser)
- int [input_open](#) (char *filename)

Variables

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

4.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.7.2 Function Documentation

4.7.2.1 input_error()

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

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

Parameters

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

Definition at line [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.

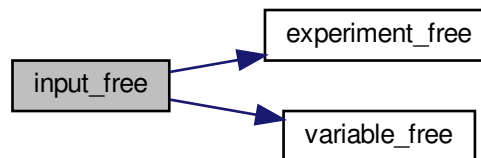
Definition at line [81](#) of file [input.c](#).

```

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");

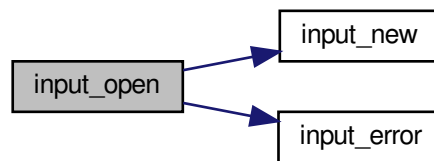
```

```

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()

```

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

```

Function to open the input file in JSON format.

Returns

1_on_success, 0_on_error.

Parameters

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

Definition at line 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,
LABEL_SEED,
                                DEFAULT_RANDOM_SEED, &error_code);
00628
00629 if (error_code)
00630 {
00631     input_error (_("Bad pseudo-random numbers generator seed"));
00632     goto exit_on_error;
00633 }
00634
00635 // Opening algorithm
00636 buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00637 if (!strcmp (buffer, LABEL_MONTE_CARLO))
00638 {
00639     input->algorithm = ALGORITHM_MONTE_CARLO;
00640
00641     // Obtaining simulations number
00642     input->nsimulations
00643 = 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,
LABEL_NPOPULATION, &error_code);
00663         if (error_code || input->nsimulations < 3)

```

```

00664         {
00665             input_error (_("Invalid population number"));
00666             goto exit_on_error;
00667         }
00668     }
00669     else
00670     {
00671         input_error (_("No population number"));
00672         goto exit_on_error;
00673     }
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
00693     // Obtaining mutation probability
00694     if (json_object_get_member (object, LABEL_MUTATION))
00695     {
00696         input->mutation_ratio
00697         = json_object_get_float (object, LABEL_MUTATION, &error_code
00698 );
00699         if (error_code || input->mutation_ratio < 0.
00700             || input->mutation_ratio >= 1.)
00701         {
00702             input_error (_("Invalid mutation probability"));
00703             goto exit_on_error;
00704         }
00705     }
00706     else
00707     {
00708         input_error (_("No mutation probability"));
00709         goto exit_on_error;
00710     }
00711
00712     // Obtaining reproduction probability
00713     if (json_object_get_member (object, LABEL_REPRODUCTION))
00714     {
00715         input->reproduction_ratio
00716         = json_object_get_float (object,
00717 LABEL_REPRODUCTION, &error_code);
00718         if (error_code || input->reproduction_ratio < 0.
00719             || input->reproduction_ratio >= 1.0)
00720         {
00721             input_error (_("Invalid reproduction probability"));
00722             goto exit_on_error;
00723         }
00724     }
00725     else
00726     {
00727         input_error (_("No reproduction probability"));
00728         goto exit_on_error;
00729     }
00730
00731     // Obtaining adaptation probability
00732     if (json_object_get_member (object, LABEL_ADAPTATION))
00733     {
00734         input->adaptation_ratio
00735         = json_object_get_float (object,
00736 LABEL_ADAPTATION, &error_code);
00737         if (error_code || input->adaptation_ratio < 0.
00738             || input->adaptation_ratio >= 1.)
00739         {
00740             input_error (_("Invalid adaptation probability"));
00741             goto exit_on_error;
00742         }
00743     }
00744     else
00745     {
00746         input_error (_("No adaptation probability"));
00747         goto exit_on_error;
00748     }
00749
00750     // 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
00779     // Obtaining best number
00780     input->nbest
00781         = json_object_get_uint_with_default (object,
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
00789     // Obtaining tolerance
00790     input->tolerance
00791         = json_object_get_float_with_default (object,
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
00799     // Getting hill climbing method parameters
00800     if (json_object_get_member (object, LABEL_NSTEPS))
00801     {
00802         input->nsteps
00803             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00804         if (error_code)
00805         {
00806             input_error (_("Invalid steps number"));
00807             goto exit_on_error;
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,
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     }
00824     else
00825     {
00826         input_error (_("Unknown method to estimate the hill climbing"));
00827         goto exit_on_error;
00828     }

```

```

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",
input->nexperiments);
00859 #endif
00860         child = json_array_get_element (array, i);
00861         if (!input->nexperiments)
00862         {
00863             if (!experiment_open_json (input->experiment, child, 0))
00864                 goto exit_on_error;
00865         }
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",
input->nexperiments);
00876 #endif
00877 }
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,
00918 LABEL_P, &error_code);
00919         if (!error_code)
00920         {
00921             input_error (_("Bad P parameter"));
00922             goto exit_on_error;
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()

```

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

```

Function to open the input file in XML format.

Returns

1_on_success, 0_on_error.

Parameters

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

Definition at line 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
00192     // Opening evaluator program name
00193     input->evaluator =
00194         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196     // Obtaining pseudo-random numbers generator seed
00197     input->seed
00198         = xml_node_get_uint_with_default (node, (const xmlChar *)
00199 LABEL_SEED,
00200                                     DEFAULT_RANDOM_SEED, &error_code);
00201     if (error_code)
00202     {
00203         input_error (_("Bad pseudo-random numbers generator seed"));
00204         goto exit_on_error;
00205     }
00206
00207     // Opening algorithm
00208     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 }
00359 // Obtaining best number
00360 input->nbest
00361 = xml_node_get_uint_with_default (node, (const xmlChar *)
00362 LABEL_NBEST,
00363                                   1, &error_code);
00364 if (error_code || !input->nbest)
00365 {
00366     input_error (_("Invalid best number"));
00367     goto exit_on_error;
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 }

```



```

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 *)
00401                                     LABEL_NESTIMATES,
00402                                     &error_code);
00403             if (error_code || !input->nestimates)
00404             {
00405                 input_error (_("Invalid estimates number"));
00406                 goto exit_on_error;
00407             }
00408         }
00409         else
00410         {
00411             input_error (_("Unknown method to estimate the hill climbing"));
00412             goto exit_on_error;
00413         }
00414         xmlFree (buffer);
00415         buffer = NULL;
00416         input->relaxation
00417             = xml_node_get_float_with_default (node,
00418                                               (const xmlChar *)
00419                                               LABEL_RELAXATION,
00420                                               DEFAULT_RELAXATION, &error_code);
00421         if (error_code || input->relaxation < 0. || input->
00422             relaxation > 2.)
00423         {
00424             input_error (_("Invalid relaxation parameter"));
00425             goto exit_on_error;
00426         }
00427         else
00428             input->nsteps = 0;
00429         // Obtaining the threshold
00430         input->threshold =
00431             xml_node_get_float_with_default (node, (const xmlChar *)
00432                                             LABEL_THRESHOLD,
00433                                             0., &error_code);
00434         if (error_code)
00435         {
00436             input_error (_("Invalid threshold"));
00437             goto exit_on_error;
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 +

```

```

    input->nexperiments,
00459         child, input->experiment->
ninputs))
00460         goto exit_on_error;
00461     }
00462     ++input->nexperiments;
00463 #if DEBUG_INPUT
00464     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00465             input->nexperiments);
00466 #endif
00467 }
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->nvariables, child,
input->algorithm, input->nsteps))
00495         goto exit_on_error;
00496     if (input->algorithm == ALGORITHM_SWEEP
00497         || input->algorithm == ALGORITHM_ORTHOGONAL)
00498         input->nsimulations *= input->variable[
input->nvariables].nsweeps;
00500     ++input->nvariables;
00501 }
00502 if (!input->nvariables)
00503 {
00504     input_error (_("No optimization variables"));
00505     goto exit_on_error;
00506 }
00507 if (input->nbest > input->nsimulations)
00508 {
00509     input_error (_("Best number higher than simulations number"));
00510     goto exit_on_error;
00511 }
00512 buffer = NULL;
00513
00514 // Obtaining the error norm
00515 if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00516 {
00517     buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00518     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00519         input->norm = ERROR_NORM_EUCLIDIAN;
00520     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00521         input->norm = ERROR_NORM_MAXIMUM;
00522     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>

```

```

00047 #include "utils.h"
00048 #include "experiment.h"
00049 #include "variable.h"
00050 #include "input.h"
00051
00052 #define DEBUG_INPUT 0
00053
00054 Input input[1];
00055
00056 const char *result_name = "result";
00057 const char *variables_name = "variables";
00058
00062 void
00063 input_new ()
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 }
00076
00080 void
00081 input_free ()
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 }
00114
00118 static void
00119 input_error (char *message)
00120 {
00121     char buffer[64];
00122     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00123     error_message = g_strdup (buffer);
00124 }
00125
00131 static inline 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 *)
LABEL_SEED,
                                DEFAULT_RANDOM_SEED, &error_code);
00198
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,
                                &error_code);
00214
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
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->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
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
00359     // Obtaining best number
00360     input->nbest
00361         = xml_node_get_uint_with_default (node, (const xmlChar *)
00362 LABEL_NBEST,
00363                                         1, &error_code);
00364     if (error_code || !input->nbest)
00365     {
00366         input_error (_("Invalid best number"));
00367         goto exit_on_error;
00368     }
00369
00370     // Obtaining tolerance
00371     input->tolerance
00372         = xml_node_get_float_with_default (node,
00373                                         (const xmlChar *) LABEL_TOLERANCE,
00374                                         0., &error_code);
00375     if (error_code || input->tolerance < 0.)
00376     {
00377         input_error (_("Invalid tolerance"));
00378         goto exit_on_error;
00379     }
00380
00381     // Getting hill climbing method parameters
00382     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00383     {
00384         input->nsteps =
00385             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00386                             &error_code);
00387         if (error_code)
00388         {
00389             input_error (_("Invalid steps number"));
00390             goto exit_on_error;
00391         }
00392     }
00393
00394     #if DEBUG_INPUT
00395     fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00396     #endif
00397
00398     buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
00399     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00400         input->climbing = CLIMBING_METHOD_COORDINATES;
00401     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00402     {
00403         input->climbing = CLIMBING_METHOD_RANDOM;
00404         input->nestimates
00405             = xml_node_get_uint (node, (const xmlChar *)
00406 LABEL_NESTIMATES,
00407                                 &error_code);
00408         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 *)
00418                                           LABEL_RELAXATION,
00419                                           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 }
00426     else
00427         input->nsteps = 0;
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 // 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 }
00467     if (!input->nexperiments)
00468     {
00469         input_error (_("No optimization experiments"));
00470         goto exit_on_error;
00471     }
00472     buffer = NULL;
00473 // Reading the variables data
00474 if (input->algorithm == ALGORITHM_SWEEP
00475     || input->algorithm == ALGORITHM_ORTHOGONAL)
00476     input->nsimulations = 1;
00477 for (; child; child = child->next)
00478 {
00479 #if DEBUG_INPUT
00480     fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00481 #endif
00482     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00483     {
00484         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->
nvariables, child,
00495         input->algorithm, input->nsteps))
00496         goto exit_on_error;
00497     if (input->algorithm == ALGORITHM_SWEEP
00498         || input->algorithm == ALGORITHM_ORTHOGONAL)
00499         input->nsimulations *= input->variable[input->
nvariables].nsweeps;
00500     ++input->nvariables;
00501 }
00502 if (!input->nvariables)
00503 {
00504     input_error (_("No optimization variables"));
00505     goto exit_on_error;
00506 }
00507 if (input->nbest > input->nsimulations)
00508 {
00509     input_error (_("Best number higher than simulations number"));
00510     goto exit_on_error;
00511 }
00512 buffer = NULL;
00513 // Obtaining the error norm
00514 if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00515 {
00516     buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00517     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00518         input->norm = ERROR_NORM_EUCLIDIAN;
00519     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00520         input->norm = ERROR_NORM_MAXIMUM;
00521     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00522     {
00523         input->norm = ERROR_NORM_P;
00524         input->p
00525             = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00526         if (error_code)
00527         {
00528             input_error (_("Bad P parameter"));
00529             goto exit_on_error;
00530         }
00531     }
00532     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00533         input->norm = ERROR_NORM_TAXICAB;
00534     else
00535     {
00536         input_error (_("Unknown error norm"));
00537         goto exit_on_error;
00538     }
00539     xmlFree (buffer);
00540 }
00541 else
00542     input->norm = ERROR_NORM_EUCLIDIAN;
00543 // Closing the XML document
00544 xmlFreeDoc (doc);
00545 #if DEBUG_INPUT
00546 fprintf (stderr, "input_open_xml: end\n");
00547 #endif
00548 return 1;
00549
00550 exit_on_error:
00551 xmlFree (buffer);
00552 xmlFreeDoc (doc);
00553 #if DEBUG_INPUT
00554 fprintf (stderr, "input_open_xml: end\n");
00555 #endif
00556 return 0;
00557 }
00558
00559 static inline int
00560 input_open_json (JsonParser * parser)
00561 {
00562     JsonNode *node, *child;
00563     JsonObject *object;
00564     JsonArray *array;
00565     const char *buffer;
00566     int error_code;
00567     unsigned int i, n;
00568 }

```

```

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);
00629     if (error_code)
00630     {
00631         input_error (_("Bad pseudo-random numbers generator seed"));
00632         goto exit_on_error;
00633     }
00634
00635     // Opening algorithm
00636     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00637     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00638     {
00639         input->algorithm = ALGORITHM_MONTE_CARLO;
00640
00641         // Obtaining simulations number
00642         input->nsimulations
00643         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00644 );
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     else
00670     {
00671         input_error (_("No population number"));
00672         goto exit_on_error;
00673     }
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     else
00687     {
00688         input_error (_("No generations number"));
00689         goto exit_on_error;
00690     }
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     else
00705     {
00706         input_error (_("No mutation probability"));
00707         goto exit_on_error;
00708     }
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     else
00723     {
00724         input_error (_("No reproduction probability"));
00725         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     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->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
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;
00818             input->nestimates
00819             = json_object_get_uint (object,
00820 LABEL_NESTIMATES, &error_code);
00821             if (error_code || !input->nestimates)
00822             {
00823                 input_error (_("Invalid estimates number"));
00824                 goto exit_on_error;
00825             }
00826         }
00827     }
00828     else
00829     {
00830         input_error (_("Unknown method to estimate the hill climbing"));
00831         goto exit_on_error;
00832     }

```

```

00827     }
00828     input->relaxation
00829     = json_object_get_float_with_default (object,
00830     LABEL_RELAXATION,
00831     DEFAULT_RELAXATION,
00832     &error_code);
00833     if (error_code || input->relaxation < 0. || input->
00834     relaxation > 2.)
00835     {
00836         input_error (_("Invalid relaxation parameter"));
00837         goto exit_on_error;
00838     }
00839     else
00840     {
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             #if DEBUG_INPUT
00878             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00879             input->nexperiments);
00880             #endif
00881             if (!input->nexperiments)
00882             {
00883                 input_error (_("No optimization experiments"));
00884                 goto exit_on_error;
00885             }
00886             // Reading the variables data
00887             array = json_object_get_array_member (object, LABEL_VARIABLES);
00888             n = json_array_get_length (array);
00889             input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00890             for (i = 0; i < n; ++i)
00891             {
00892                 #if DEBUG_INPUT
00893                 fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00894                 #endif
00895                 child = json_array_get_element (array, i);
00896                 if (!variable_open_json (input->variable + input->
00897                 nvariables, child,
00898                 input->algorithm, input->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))

```

```

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,
00918 LABEL_P, &error_code);
00919                 if (!error_code)
00920                     {
00921                         input_error (_("Bad P parameter"));
00922                         goto exit_on_error;
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                 parser = json_parser_new ();
00976                 if (!json_parser_load_from_file (parser, filename, NULL))
00977                     {
00978                         input_error (_("Unable to parse the input file"));
00979                         goto exit_on_error;
00980                     }
00981                 if (!input_open_json (parser))
00982                     goto exit_on_error;
00983             }
00984         else if (!input_open_xml (doc))
00985             goto exit_on_error;
00986
00987         // Getting the working directory
00988         input->directory = g_path_get_dirname (filename);
00989         input->name = g_path_get_basename (filename);
00990
00991         #if DEBUG_INPUT
00992             fprintf (stderr, "input_open: end\n");
00993         #endif
00994     }

```

```

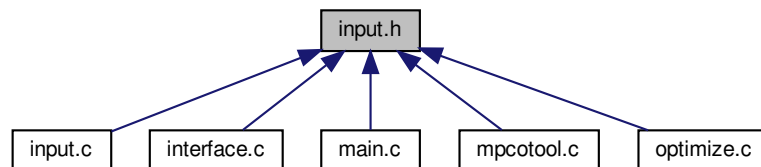
00999     return 1;
01000
01001 exit_on_error:
01002     show_error (error_message);
01003     g_free (error_message);
01004     input_free ();
01005 #if DEBUG_INPUT
01006     fprintf (stderr, "input_open: end\n");
01007 #endif
01008     return 0;
01009 }

```

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](#) ()
- 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_free()

```
void input_free ( )
```

Function to free the memory of the input file data.

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

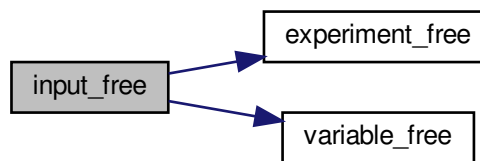
```
00082 {
00083     unsigned int i;
00084     #if DEBUG_INPUT
00085         fprintf (stderr, "input_free: start\n");
00086     #endif
00087     g_free (input->name);
00088     g_free (input->directory);
00089     for (i = 0; i < input->nexperiments; ++i)
00090         experiment_free (input->experiment + 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:



4.9.3.2 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 }

```

4.9.3.3 input_open()

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

Function to open the input file.

Returns

1_on_success, 0_on_error.

Parameters

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

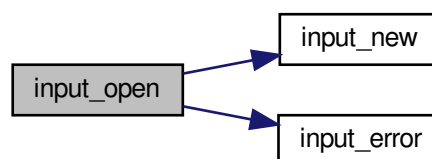
Definition at line 957 of file [input.c](#).

```

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:



4.10 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

```

```

00105 void input_new ();
00106 void input_free ();
00107 int input_open (char *filename);
00108
00109 #endif

```

4.11 interface.c File Reference

Source file to define the graphical interface functions.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "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

- static void `input_save_climbing_xml` (xmlNode *node)
- static void `input_save_climbing_json` (JsonNode *node)
- static void `input_save_xml` (xmlDoc *doc)
- static void `input_save_json` (JsonGenerator *generator)
- static void `input_save` (char *filename)
- static void `options_new` ()
- static void `running_new` ()

- static unsigned int [window_get_algorithm](#) ()
- static unsigned int [window_get_climbing](#) ()
- static unsigned int [window_get_norm](#) ()
- static void [window_save_climbing](#) ()
- static int [window_save](#) ()
- static void [window_run](#) ()
- static void [window_help](#) ()
- static void [window_about](#) ()
- static void [window_update_climbing](#) ()
- static void [window_update](#) ()
- static void [window_set_algorithm](#) ()
- static void [window_set_experiment](#) ()
- static void [window_remove_experiment](#) ()
- static void [window_add_experiment](#) ()
- static void [window_name_experiment](#) ()
- static void [window_weight_experiment](#) ()
- static void [window_inputs_experiment](#) ()
- static void [window_template_experiment](#) (void *data)
- static void [window_set_variable](#) ()
- static void [window_remove_variable](#) ()
- static void [window_add_variable](#) ()
- static void [window_label_variable](#) ()
- static void [window_precision_variable](#) ()
- static void [window_rangemin_variable](#) ()
- static void [window_rangemax_variable](#) ()
- static void [window_rangeminabs_variable](#) ()
- static void [window_rangemaxabs_variable](#) ()
- static void [window_step_variable](#) ()
- static void [window_update_variable](#) ()
- static int [window_read](#) (char *filename)
- static void [window_open](#) ()
- void [window_new](#) (GtkApplication *application)

Variables

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

4.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.11.2 Function Documentation

4.11.2.1 input_save()

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

Function to save the input file.

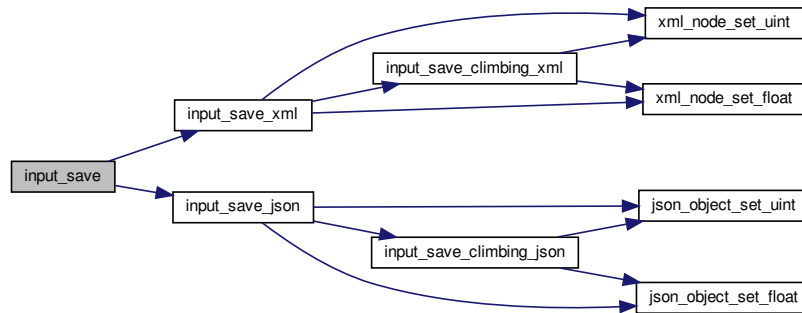
Parameters

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

Definition at line 585 of file [interface.c](#).

```
00586 {
00587     xmlDoc *doc;
00588     JsonGenerator *generator;
00589
00590 #if DEBUG_INTERFACE
00591     fprintf (stderr, "input_save: start\n");
00592 #endif
00593
00594     // Getting the input file directory
00595     input->name = g_path_get_basename (filename);
00596     input->directory = g_path_get_dirname (filename);
00597
00598     if (input->type == INPUT_TYPE_XML)
00599     {
00600         // Opening the input file
00601         doc = xmlNewDoc ((const xmlChar *) "1.0");
00602         input_save_xml (doc);
00603
00604         // Saving the XML file
00605         xmlSaveFormatFile (filename, doc, 1);
00606
00607         // Freeing memory
00608         xmlFreeDoc (doc);
00609     }
00610     else
00611     {
00612         // Opening the input file
00613         generator = json_generator_new ();
00614         json_generator_set_pretty (generator, TRUE);
00615         input_save_json (generator);
00616
00617         // Saving the JSON file
00618         json_generator_to_file (generator, filename, NULL);
00619
00620         // Freeing memory
00621         g_object_unref (generator);
00622     }
00623
00624 #if DEBUG_INTERFACE
00625     fprintf (stderr, "input_save: end\n");
00626 #endif
00627 }
```

Here is the call graph for this function:



4.11.2.2 input_save_climbing_json()

```
static void input_save_climbing_json (
    JsonNode * node ) [static]
```

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

Parameters

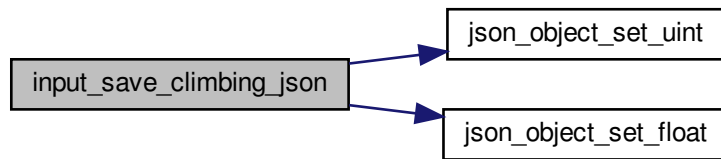
<i>node</i>	JSON node.
-------------	------------

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

```

00203 {
00204     JsonObject *object;
00205     #if DEBUG_INTERFACE
00206     fprintf (stderr, "input_save_climbing_json: start\n");
00207     #endif
00208     object = json_node_get_object (node);
00209     if (input->nsteps)
00210     {
00211         json_object_set_uint (object, LABEL_NSTEPS,
00212             input->nsteps);
00213         if (input->relaxation != DEFAULT_RELAXATION)
00214             json_object_set_float (object, LABEL_RELAXATION,
00215                 input->relaxation);
00216         switch (input->climbing)
00217         {
00218             case CLIMBING_METHOD_COORDINATES:
00219                 json_object_set_string_member (object, LABEL_CLIMBING,
00220                     LABEL_COORDINATES);
00221                 break;
00222             default:
00223                 json_object_set_string_member (object, LABEL_CLIMBING,
00224                     LABEL_RANDOM);
00225                 json_object_set_uint (object, LABEL_NESTIMATES,
00226                     input->nestimates);
00227         }
00228     }
00229     #if DEBUG_INTERFACE
00230     fprintf (stderr, "input_save_climbing_json: end\n");
00231     #endif
00232 }
```


Here is the call graph for this function:



4.11.2.3 input_save_climbing_xml()

```
static void input_save_climbing_xml (
    xmlNode * node ) [static]
```

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

Parameters

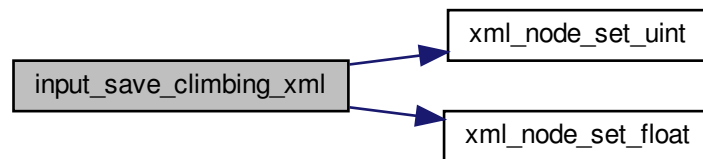
<i>node</i>	XML node.
-------------	-----------

Definition at line 169 of file [interface.c](#).

```

00170 {
00171     #if DEBUG_INTERFACE
00172         fprintf (stderr, "input_save_climbing_xml: start\n");
00173     #endif
00174     if (input->nsteps)
00175     {
00176         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00177             input->nsteps);
00178         if (input->relaxation != DEFAULT_RELAXATION)
00179             xml_node_set_float (node, (const xmlChar *)
00180                 LABEL_RELAXATION,
00181                 input->relaxation);
00182         switch (input->climbing)
00183         {
00184             case CLIMBING_METHOD_COORDINATES:
00185                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00186                     (const xmlChar *) LABEL_COORDINATES);
00187                 break;
00188             default:
00189                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
00190                     (const xmlChar *) LABEL_RANDOM);
00191                 xml_node_set_uint (node, (const xmlChar *)
00192                     LABEL_NESTIMATES,
00193                     input->nestimates);
00194         }
00195     }
00196     #if DEBUG_INTERFACE
00197         fprintf (stderr, "input_save_climbing_xml: end\n");
00198     #endif
00199 }
```

Here is the call graph for this function:



4.11.2.4 input_save_json()

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

Function to save the input file in JSON format.

Parameters

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

Definition at line 414 of file [interface.c](#).

```

00415 {
00416     unsigned int i, j;
00417     char *buffer;
00418     JsonNode *node, *child;
00419     JsonObject *object;
00420     JsonArray *array;
00421     GFile *file, *file2;
00422
00423     #if DEBUG_INTERFACE
00424         fprintf (stderr, "input_save_json: start\n");
00425     #endif
00426
00427     // Setting root JSON node
00428     node = json_node_new (JSON_NODE_OBJECT);
00429     object = json_node_get_object (node);
00430     json_generator_set_root (generator, node);
00431
00432     // Adding properties to the root JSON node
00433     if (strcmp (input->result, result_name))
00434         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00435     if (strcmp (input->variables, variables_name))
00436         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00437
00438     file = g_file_new_for_path (input->directory);
00439     file2 = g_file_new_for_path (input->simulator);
00440     buffer = g_file_get_relative_path (file, file2);
00441     g_object_unref (file2);
00442     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00443     g_free (buffer);
00444     if (input->evaluator)
00445     {
00446         file2 = g_file_new_for_path (input->evaluator);
00447         buffer = g_file_get_relative_path (file, file2);

```

```

00448     g_object_unref (file2);
00449     if (strlen (buffer))
00450         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00451     g_free (buffer);
00452 }
00453 if (input->seed != DEFAULT_RANDOM_SEED)
00454     json_object_set_uint (object, LABEL_SEED,
00455 input->seed);
00455
00456 // Setting the algorithm
00457 buffer = (char *) g_slice_alloc (64);
00458 switch (input->algorithm)
00459 {
00460     case ALGORITHM_MONTE_CARLO:
00461         json_object_set_string_member (object, LABEL_ALGORITHM,
00462 LABEL_MONTE_CARLO);
00463         snprintf (buffer, 64, "%u", input->nsimulations);
00464         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00465         snprintf (buffer, 64, "%u", input->niterations);
00466         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00467         snprintf (buffer, 64, "%.3lg", input->tolerance);
00468         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00469         snprintf (buffer, 64, "%u", input->nbest);
00470         json_object_set_string_member (object, LABEL_NBEST, buffer);
00471         input_save_climbing_json (node);
00472         break;
00473     case ALGORITHM_SWEEP:
00474         json_object_set_string_member (object, LABEL_ALGORITHM,
00475 LABEL_SWEEP);
00476         snprintf (buffer, 64, "%u", input->niterations);
00477         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00478         snprintf (buffer, 64, "%.3lg", input->tolerance);
00479         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00480         snprintf (buffer, 64, "%u", input->nbest);
00481         json_object_set_string_member (object, LABEL_NBEST, buffer);
00482         input_save_climbing_json (node);
00483         break;
00484     case ALGORITHM_ORTHOGONAL:
00485         json_object_set_string_member (object, LABEL_ALGORITHM,
00486 LABEL_ORTHOGONAL);
00487         snprintf (buffer, 64, "%u", input->niterations);
00488         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00489         snprintf (buffer, 64, "%.3lg", input->tolerance);
00490         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00491         snprintf (buffer, 64, "%u", input->nbest);
00492         json_object_set_string_member (object, LABEL_NBEST, buffer);
00493         input_save_climbing_json (node);
00494         break;
00495     default:
00496         json_object_set_string_member (object, LABEL_ALGORITHM,
00497 LABEL_GENETIC);
00498         snprintf (buffer, 64, "%u", input->nsimulations);
00499         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00500         printf (buffer, 64, "%u", input->niterations);
00501         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00502         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00503         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00504         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00505         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00506         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00507         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00508         break;
00509 }
00510 g_slice_free1 (64, buffer);
00511 if (input->threshold != 0.)
00512     json_object_set_float (object, LABEL_THRESHOLD,
00513 input->threshold);
00514
00515 // Setting the experimental data
00516 array = json_array_new ();
00517 for (i = 0; i < input->nexperiments; ++i)
00518 {
00519     child = json_node_new (JSON_NODE_OBJECT);
00520     object = json_node_get_object (child);
00521     json_object_set_string_member (object, LABEL_NAME,
00522 input->experiment[i].name);
00523     if (input->experiment[i].weight != 1.)
00524         json_object_set_float (object, LABEL_WEIGHT,
00525 input->experiment[i].weight);
00526     for (j = 0; j < input->experiment->ninputs; ++j)
00527         json_object_set_string_member (object, stencil[j],
00528 input->experiment[i].
00529 stencil[j]);
00530     json_array_add_element (array, child);
00531 }
00532 json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00533

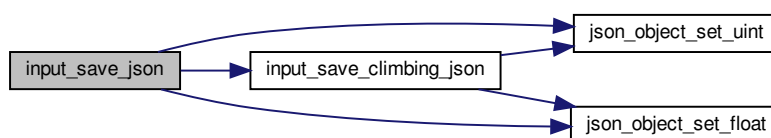
```

```

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:
00570         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00571         break;
00572     case ERROR_NORM_P:
00573         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00574         json_object_set_float (object, LABEL_P, input->
00575                                p);
00576         break;
00577     case ERROR_NORM_TAXICAB:
00578         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00579 }
00580 #if DEBUG_INTERFACE
00581 fprintf (stderr, "input_save_json: end\n");
00582 #endif
00583 }

```

Here is the call graph for this function:



4.11.2.5 input_save_xml()

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

Function to save the input file in XML format.

Parameters

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

Definition at line 234 of file [interface.c](#).

```
00235 {
00236     unsigned int i, j;
00237     char *buffer;
00238     xmlNode *node, *child;
00239     GFile *file, *file2;
00240
00241     #if DEBUG_INTERFACE
00242     fprintf (stderr, "input_save_xml: start\n");
00243     #endif
00244
00245     // Setting root XML node
00246     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00247     xmlDocSetRootElement (doc, node);
00248
00249     // Adding properties to the root XML node
00250     if (xmlStrcmp
00251         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00252         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00253                     (xmlChar *) input->result);
00254     if (xmlStrcmp
00255         ((const xmlChar *) input->variables, (const xmlChar *)
00256         variables_name))
00257         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00258                     (xmlChar *) input->variables);
00259     file = g_file_new_for_path (input->directory);
00260     file2 = g_file_new_for_path (input->simulator);
00261     buffer = g_file_get_relative_path (file, file2);
00262     g_object_unref (file2);
00263     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00264     g_free (buffer);
00265     if (input->evaluator)
00266     {
00267         file2 = g_file_new_for_path (input->evaluator);
00268         buffer = g_file_get_relative_path (file, file2);
00269         g_object_unref (file2);
00270         if (xmlStrlen ((xmlChar *) buffer))
00271             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00272                         (xmlChar *) buffer);
00273         g_free (buffer);
00274     }
00275     if (input->seed != DEFAULT_RANDOM_SEED)
00276         xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
00277         input->seed);
00278
00279     // Setting the algorithm
00280     buffer = (char *) g_slice_alloc (64);
00281     switch (input->algorithm)
00282     {
00283     case ALGORITHM_MONTE_CARLO:
00284         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00285                     (const xmlChar *) LABEL_MONTE_CARLO);
00286         snprintf (buffer, 64, "%u", input->nsimulations);
00287         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00288                     (xmlChar *) buffer);
00289         snprintf (buffer, 64, "%u", input->niterations);
00290         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00291                     (xmlChar *) buffer);
00292         snprintf (buffer, 64, "%.3lg", input->tolerance);
00293         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00294         snprintf (buffer, 64, "%u", input->nbest);
00295         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00296         input_save_climbing_xml (node);
00297         break;
00298     case ALGORITHM_SWEEP:
```

```

00297     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00298                 (const xmlChar *) LABEL_SWEEP);
00299     snprintf (buffer, 64, "%u", input->niterations);
00300     xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00301                 (xmlChar *) buffer);
00302     snprintf (buffer, 64, "%.3lg", input->tolerance);
00303     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00304     snprintf (buffer, 64, "%u", input->nbest);
00305     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00306     input_save_climbing_xml (node);
00307     break;
00308 case ALGORITHM_ORTHOGONAL:
00309     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00310                 (const xmlChar *) LABEL_ORTHOGONAL);
00311     snprintf (buffer, 64, "%u", input->niterations);
00312     xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00313                 (xmlChar *) buffer);
00314     snprintf (buffer, 64, "%.3lg", input->tolerance);
00315     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00316     snprintf (buffer, 64, "%u", input->nbest);
00317     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00318     input_save_climbing_xml (node);
00319     break;
00320 default:
00321     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00322                 (const xmlChar *) LABEL_GENETIC);
00323     snprintf (buffer, 64, "%u", input->nsimulations);
00324     xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00325                 (xmlChar *) buffer);
00326     snprintf (buffer, 64, "%u", input->niterations);
00327     xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00328                 (xmlChar *) buffer);
00329     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00330     xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00331     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00332     xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00333                 (xmlChar *) buffer);
00334     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00335     xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00336     break;
00337 }
00338 g_slice_free1 (64, buffer);
00339 if (input->threshold != 0.)
00340     xml_node_set_float (node, (const xmlChar *)
00341 LABEL_THRESHOLD,
00342                         input->threshold);
00343 // Setting the experimental data
00344 for (i = 0; i < input->nexperiments; ++i)
00345 {
00346     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00347     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00348                 (xmlChar *) input->experiment[i].name);
00349     if (input->experiment[i].weight != 1.)
00350         xml_node_set_float (child, (const xmlChar *)
00351 LABEL_WEIGHT,
00352                             input->experiment[i].weight);
00353     for (j = 0; j < input->experiment->ninputs; ++j)
00354         xmlSetProp (child, (const xmlChar *) stencil[j],
00355                     (xmlChar *) input->experiment[i].stencil[j]);
00356 }
00357 // Setting the variables data
00358 for (i = 0; i < input->nvariables; ++i)
00359 {
00360     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00361     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00362                 (xmlChar *) input->variable[i].name);
00363     xml_node_set_float (child, (const xmlChar *)
00364 LABEL_MINIMUM,
00365                         input->variable[i].rangemin);
00366     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00367         xml_node_set_float (child, (const xmlChar *)
00368 LABEL_ABSOLUTE_MINIMUM,
00369                             input->variable[i].rangeminabs);
00370     xml_node_set_float (child, (const xmlChar *)
00371 LABEL_MAXIMUM,
00372                         input->variable[i].rangemax);
00373     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00374         xml_node_set_float (child, (const xmlChar *)
00375 LABEL_ABSOLUTE_MAXIMUM,
00376                             input->variable[i].rangemaxabs);
00377     if (input->variable[i].precision !=
00378         DEFAULT_PRECISION)
00379         xml_node_set_uint (child, (const xmlChar *)
00380 LABEL_PRECISION,
00381                         input->variable[i].precision);

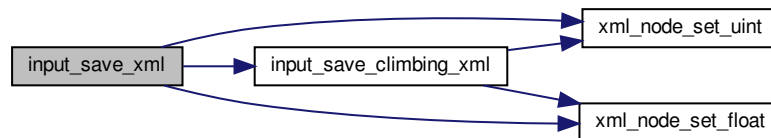
```

```

00376     if (input->algorithm == ALGORITHM_SWEEP
00377         || input->algorithm == ALGORITHM_ORTHOGONAL)
00378         xml_node_set_uint (child, (const xmlChar *)
LABEL_NSWEEPS,
00379                             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);
00383     if (input->nsteps)
00384         xml_node_set_float (child, (const xmlChar *)
LABEL_STEP,
00385                             input->variable[i].step);
00386 }
00387
00388 // Saving the error norm
00389 switch (input->norm)
00390 {
00391     case ERROR_NORM_MAXIMUM:
00392         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00393                     (const xmlChar *) LABEL_MAXIMUM);
00394         break;
00395     case ERROR_NORM_P:
00396         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397                     (const xmlChar *) LABEL_P);
00398         xml_node_set_float (node, (const xmlChar *) LABEL_P,
input->p);
00399         break;
00400     case ERROR_NORM_TAXICAB:
00401         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00402                     (const xmlChar *) LABEL_TAXICAB);
00403 }
00404
00405 #if DEBUG_INTERFACE
00406 fprintf (stderr, "input_save: end\n");
00407 #endif
00408 }

```

Here is the call graph for this function:



4.11.2.6 options_new()

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

Function to open the options dialog.

Definition at line 633 of file [interface.c](#).

```

00634 {
00635     #if DEBUG_INTERFACE
00636         fprintf (stderr, "options_new: start\n");
00637     #endif
00638     options->label_seed = (GtkLabel *)
00639         gtk_label_new (_("Pseudo-random numbers generator seed"));
00640     options->spin_seed = (GtkSpinButton *)
00641         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00642     gtk_widget_set_tooltip_text

```

```

00643     (GTK_WIDGET (options->spin_seed),
00644     _("Seed to init the pseudo-random numbers generator"));
00645     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00646     options->label_threads = (GtkLabel *)
00647     gtk_label_new (_("Threads number for the stochastic algorithm"));
00648     options->spin_threads
00649     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00650     gtk_widget_set_tooltip_text
00651     (GTK_WIDGET (options->spin_threads),
00652     _("Number of threads to perform the calibration/optimization for "
00653     "the stochastic algorithm"));
00654     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00655     options->label_climbing = (GtkLabel *)
00656     gtk_label_new (_("Threads number for the hill climbing method"));
00657     options->spin_climbing =
00658     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00659     gtk_widget_set_tooltip_text
00660     (GTK_WIDGET (options->spin_climbing),
00661     _("Number of threads to perform the calibration/optimization for the "
00662     "hill climbing method"));
00663     gtk_spin_button_set_value (options->spin_climbing,
00664     (gdouble) nthreads_climbing);
00665     options->grid = (GtkGrid *) gtk_grid_new ();
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_seed), 0, 0, 1, 1);
00667     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_seed), 1, 0, 1, 1);
00668     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_threads),
00669     0, 1, 1, 1);
00670     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_threads),
00671     1, 1, 1, 1);
00672     gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_climbing), 0, 2, 1,
00673     1);
00674     gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_climbing), 1, 2, 1,
00675     1);
00676     gtk_widget_show_all (GTK_WIDGET (options->grid));
00677     options->dialog = (GtkDialog *)
00678     gtk_dialog_new_with_buttons (_("Options"),
00679     window->window,
00680     GTK_DIALOG_MODAL,
00681     _("_OK"), GTK_RESPONSE_OK,
00682     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00683     gtk_container_add
00684     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00685     GTK_WIDGET (options->grid));
00686     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00687     {
00688         input->seed
00689         = (unsigned long int) gtk_spin_button_get_value (options->
spin_seed);
00690         nthreads = gtk_spin_button_get_value_as_int (options->
spin_threads);
00691         nthreads_climbing
00692         = gtk_spin_button_get_value_as_int (options->spin_climbing);
00693     }
00694     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00695     #if DEBUG_INTERFACE
00696     fprintf (stderr, "options_new: end\n");
00697     #endif
00698 }

```

4.11.2.7 running_new()

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

Function to open the running dialog.

Definition at line 704 of file [interface.c](#).


```

00705 {
00706     #if DEBUG_INTERFACE
00707         fprintf (stderr, "running_new: start\n");
00708     #endif
00709     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00710     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00711     running->grid = (GtkGrid *) gtk_grid_new ();
00712     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00713     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00714     running->dialog = (GtkDialog *)
00715         gtk_dialog_new_with_buttons (_("Calculating"),
00716                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00717     gtk_container_add (GTK_CONTAINER
00718                       (gtk_dialog_get_content_area (running->dialog)),
00719                       GTK_WIDGET (running->grid));
00720     gtk_spinner_start (running->spinner);
00721     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00722     #if DEBUG_INTERFACE
00723         fprintf (stderr, "running_new: end\n");
00724     #endif
00725 }

```

4.11.2.8 window_about()

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

Function to show an about dialog.

Definition at line 1058 of file [interface.c](#).

```

01059 {
01060     static const gchar *authors[] = {
01061         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01062         "Borja Latorre Garcés <borja.latorre@csic.es>",
01063         NULL
01064     };
01065     #if DEBUG_INTERFACE
01066         fprintf (stderr, "window_about: start\n");
01067     #endif
01068     gtk_show_about_dialog
01069         (window->window,
01070          "program_name", "MPCOTool",
01071          "comments",
01072          _("The Multi-Purposes Calibration and Optimization Tool.\n"
01073            "A software to perform calibrations or optimizations of empirical "
01074            "parameters"),
01075          "authors", authors,
01076          "translator-credits",
01077          "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01078            "(english, french and spanish)\n"
01079            "Uğur Çayoğlu (german)",
01080          "version", "4.0.1",
01081          "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
01082          "logo", window->logo,
01083          "website", "https://github.com/jburguete/mpcotool",
01084          "license-type", GTK_LICENSE_BSD, NULL);
01085     #if DEBUG_INTERFACE
01086         fprintf (stderr, "window_about: end\n");
01087     #endif
01088 }

```

4.11.2.9 window_add_experiment()

static void window_add_experiment () [static]

Function to add an experiment in the main window.

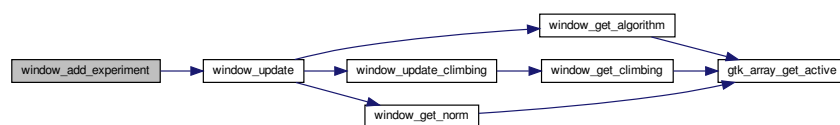
Definition at line 1393 of file [interface.c](#).

```

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

```

Here is the call graph for this function:



4.11.2.10 window_add_variable()

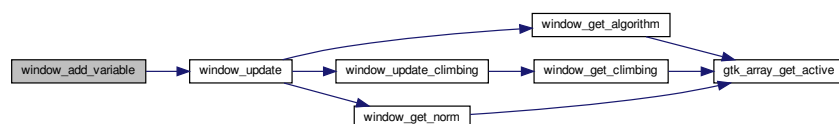
```
static void window_add_variable ( ) [static]
```

Function to add a variable in the main window.

Definition at line 1656 of file [interface.c](#).

```
01657 {
01658     unsigned int i, j;
01659     #if DEBUG_INTERFACE
01660     fprintf (stderr, "window_add_variable: start\n");
01661     #endif
01662     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01663     g_signal_handler_block (window->combo_variable, window->
01664         id_variable);
01665     gtk_combo_box_text_insert_text (window->combo_variable, i,
01666         input->variable[i].name);
01667     g_signal_handler_unblock (window->combo_variable, window->
01668         id_variable);
01669     input->variable = (Variable *) g_realloc
01670         (input->variable, (input->nvariables + 1) * sizeof (
01671         Variable));
01672     for (j = input->nvariables - 1; j > i; --j)
01673         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01674         Variable));
01675     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01676         Variable));
01677     if (input->type == INPUT_TYPE_XML)
01678         input->variable[j + 1].name
01679             = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01680     else
01681         input->variable[j + 1].name = g_strdup (input->
01682         variable[j].name);
01683     ++input->nvariables;
01684     g_signal_handler_block (window->entry_variable, window->
01685         id_variable_label);
01686     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01687     g_signal_handler_unblock (window->entry_variable, window->
01688         id_variable_label);
01689     window_update ();
01690     #if DEBUG_INTERFACE
01691     fprintf (stderr, "window_add_variable: end\n");
01692     #endif
01693 }
```

Here is the call graph for this function:



4.11.2.11 window_get_algorithm()

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

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 733 of file [interface.c](#).

```
00734 {  
00735     unsigned int i;  
00736     #if DEBUG_INTERFACE  
00737     fprintf (stderr, "window_get_algorithm: start\n");  
00738     #endif  
00739     i = gtk_array_get_active (window->button_algorithm,  
        NALGORITHMS);  
00740     #if DEBUG_INTERFACE  
00741     fprintf (stderr, "window_get_algorithm: %u\n", i);  
00742     fprintf (stderr, "window_get_algorithm: end\n");  
00743     #endif  
00744     return i;  
00745 }
```

Here is the call graph for this function:



4.11.2.12 window_get_climbing()

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

Function to get the hill climbing method number.

Returns

Hill climbing method number.

Definition at line 753 of file [interface.c](#).

```
00754 {
00755     unsigned int i;
00756     #if DEBUG_INTERFACE
00757         fprintf (stderr, "window_get_climbing: start\n");
00758     #endif
00759     i = gtk_array_get_active (window->button_climbing,
00760                             NCLIMBINGS);
00761     #if DEBUG_INTERFACE
00762         fprintf (stderr, "window_get_climbing: %u\n", i);
00763     #endif
00764     return i;
00765 }
```

Here is the call graph for this function:



4.11.2.13 window_get_norm()

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

Function to get the norm method number.

Returns

Norm method number.

Definition at line 773 of file [interface.c](#).

```
00774 {
00775     unsigned int i;
00776     #if DEBUG_INTERFACE
00777         fprintf (stderr, "window_get_norm: start\n");
00778     #endif
00779     i = gtk_array_get_active (window->button_norm,
00780                             NNORMS);
00781     #if DEBUG_INTERFACE
00782         fprintf (stderr, "window_get_norm: %u\n", i);
00783     #endif
00784     return i;
00785 }
```

Here is the call graph for this function:



4.11.2.14 window_help()

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

Function to show a help dialog.

Definition at line 1030 of file [interface.c](#).

```
01031 {
01032     char *buffer, *buffer2;
01033     #if DEBUG_INTERFACE
01034     fprintf (stderr, "window_help: start\n");
01035     #endif
01036     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01037                               _("user-manual.pdf"), NULL);
01038     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01039     g_free (buffer2);
01040     #if GTK_MINOR_VERSION >= 22
01041     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01042     #else
01043     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01044     #endif
01045     #if DEBUG_INTERFACE
01046     fprintf (stderr, "window_help: uri=%s\n", buffer);
01047     #endif
01048     g_free (buffer);
01049     #if DEBUG_INTERFACE
01050     fprintf (stderr, "window_help: end\n");
01051     #endif
01052 }
```

4.11.2.15 window_inputs_experiment()

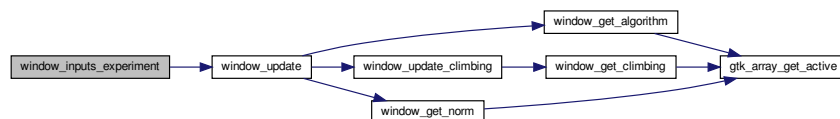
```
static void window_inputs_experiment ( ) [static]
```

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

Definition at line 1493 of file [interface.c](#).

```
01494 {
01495     unsigned int j;
01496     #if DEBUG_INTERFACE
01497     fprintf (stderr, "window_inputs_experiment: start\n");
01498     #endif
01499     j = input->experiment->ninputs - 1;
01500     if (j
01501         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01502                                           (window->check_template[j])))
01503         --input->experiment->ninputs;
01504     if (input->experiment->ninputs < MAX_NINPUTS
01505         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01506                                           (window->check_template[j])))
01507         ++input->experiment->ninputs;
01508     window_update ();
01509     #if DEBUG_INTERFACE
01510     fprintf (stderr, "window_inputs_experiment: end\n");
01511     #endif
01512 }
```

Here is the call graph for this function:



4.11.2.16 window_label_variable()

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

Function to set the variable label in the main window.

Definition at line 1691 of file [interface.c](#).

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

4.11.2.17 window_name_experiment()

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

Function to set the experiment name in the main window.

Definition at line 1446 of file [interface.c](#).

```
01447 {
01448     unsigned int i;
01449     char *buffer;
01450     GFile *file1, *file2;
01451     #if DEBUG_INTERFACE
01452     fprintf (stderr, "window_name_experiment: start\n");
01453     #endif
01454     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01455     file1
01456     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
01457         button_experiment));
01457     file2 = g_file_new_for_path (input->directory);
01458     buffer = g_file_get_relative_path (file2, file1);
01459     g_signal_handler_block (window->combo_experiment, window->
01460         id_experiment);
01460     gtk_combo_box_text_remove (window->combo_experiment, i);
01461     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01462     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01463     g_signal_handler_unblock (window->combo_experiment,
01464         window->id_experiment);
01464     g_free (buffer);
01465     g_object_unref (file2);
01466     g_object_unref (file1);
01467     #if DEBUG_INTERFACE
01468     fprintf (stderr, "window_name_experiment: end\n");
01469     #endif
01470 }
```

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 2066 of file [interface.c](#).

```

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

```



```

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

```

```

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

```

```

02297             GTK_WIDGET (window->spin_threshold));
02298 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02299 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02300 //             GTK_ALIGN_FILL);
02301
02302 // Creating the hill climbing method properties
02303 window->check_climbing = (GtkCheckButton *)
02304     gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02305 g_signal_connect (window->check_climbing, "clicked",
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",
window_update, NULL);
02312 for (i = 0; ++i < NCLIMBINGS;)
02313 {
02314     window->button_climbing[i] = (GtkRadioButton *)
02315         gtk_radio_button_new_with_mnemonic
02316             (gtk_radio_button_get_group (window->button_climbing[0]),
02317             label_climbing[i]);
02318     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02319         tip_climbing[i]);
02320     gtk_grid_attach (window->grid_climbing,
02321         GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02322     g_signal_connect (window->button_climbing[i], "clicked",
window_update,
02323         NULL);
02324 }
02325 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02326 window->spin_steps = (GtkSpinButton *)
02327     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02328 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02329 window->label_estimates
02330     = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02331 window->spin_estimates = (GtkSpinButton *)
02332     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02333 window->label_relaxation
02334     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02335 window->spin_relaxation = (GtkSpinButton *)
02336     gtk_spin_button_new_with_range (0., 2., 0.001);
02337 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_steps),
02338     0, NCLIMBINGS, 1, 1);
02339 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_steps),
02340     1, NCLIMBINGS, 1, 1);
02341 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_estimates),
02342     0, NCLIMBINGS + 1, 1, 1);
02343 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_estimates),
02344     1, NCLIMBINGS + 1, 1, 1);
02345 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_relaxation),
02346     0, NCLIMBINGS + 2, 1, 1);
02347 gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_relaxation),
02348     1, NCLIMBINGS + 2, 1, 1);
02349
02350 // Creating the array of algorithms
02351 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02352 window->button_algorithm[0] = (GtkRadioButton *)
02353     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02354 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02355     tip_algorithm[0]);
02356 gtk_grid_attach (window->grid_algorithm,
02357     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02358 g_signal_connect (window->button_algorithm[0], "clicked",
02359     window_set_algorithm, NULL);
02360 for (i = 0; ++i < NALGORITHMS;)
02361 {
02362     window->button_algorithm[i] = (GtkRadioButton *)
02363         gtk_radio_button_new_with_mnemonic
02364             (gtk_radio_button_get_group (window->button_algorithm[0]),
02365             label_algorithm[i]);
02366     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02367         tip_algorithm[i]);
02368     gtk_grid_attach (window->grid_algorithm,
02369         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02370     g_signal_connect (window->button_algorithm[i], "clicked",
02371         window_set_algorithm, NULL);
02372 }
02373 gtk_grid_attach (window->grid_algorithm,
02374     GTK_WIDGET (window->label_simulations),

```

```

02375         0, NALGORITHMS, 1, 1);
02376     gtk_grid_attach (window->grid_algorithm,
02377         GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02378     gtk_grid_attach (window->grid_algorithm,
02379         GTK_WIDGET (window->label_iterations),
02380         0, NALGORITHMS + 1, 1, 1);
02381     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02382         window->spin_iterations),
02383         1, NALGORITHMS + 1, 1, 1);
02384     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02385         window->label_tolerance),
02386         0, NALGORITHMS + 2, 1, 1);
02387     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02388         window->spin_tolerance),
02389         1, NALGORITHMS + 2, 1, 1);
02390     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02391         window->label_bests),
02392         0, NALGORITHMS + 3, 1, 1);
02393     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02394         window->spin_bests),
02395         1, NALGORITHMS + 3, 1, 1);
02396     gtk_grid_attach (window->grid_algorithm,
02397         GTK_WIDGET (window->label_population),
02398         0, NALGORITHMS + 4, 1, 1);
02399     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02400         window->spin_population),
02401         1, NALGORITHMS + 4, 1, 1);
02402     gtk_grid_attach (window->grid_algorithm,
02403         GTK_WIDGET (window->label_generations),
02404         0, NALGORITHMS + 5, 1, 1);
02405     gtk_grid_attach (window->grid_algorithm,
02406         GTK_WIDGET (window->spin_generations),
02407         1, NALGORITHMS + 5, 1, 1);
02408     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02409         window->label_mutation),
02410         0, NALGORITHMS + 6, 1, 1);
02411     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02412         window->spin_mutation),
02413         1, NALGORITHMS + 6, 1, 1);
02414     gtk_grid_attach (window->grid_algorithm,
02415         GTK_WIDGET (window->label_reproduction),
02416         0, NALGORITHMS + 7, 1, 1);
02417     gtk_grid_attach (window->grid_algorithm,
02418         GTK_WIDGET (window->spin_reproduction),
02419         1, NALGORITHMS + 7, 1, 1);
02420     gtk_grid_attach (window->grid_algorithm,
02421         GTK_WIDGET (window->label_adaptation),
02422         0, NALGORITHMS + 8, 1, 1);
02423     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02424         window->spin_adaptation),
02425         1, NALGORITHMS + 8, 1, 1);
02426     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02427         window->check_climbing),
02428         0, NALGORITHMS + 9, 2, 1);
02429     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02430         window->grid_climbing),
02431         0, NALGORITHMS + 10, 2, 1);
02432     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02433         window->label_threshold),
02434         0, NALGORITHMS + 11, 1, 1);
02435     gtk_grid_attach (window->grid_algorithm,
02436         GTK_WIDGET (window->scrolled_threshold),
02437         1, NALGORITHMS + 11, 1, 1);
02438     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02439     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02440         GTK_WIDGET (window->grid_algorithm));
02441
02442     // Creating the variable widgets
02443     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02444     gtk_widget_set_tooltip_text
02445         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02446     window->id_variable = g_signal_connect
02447         (window->combo_variable, "changed", window_set_variable, NULL);
02448     window->button_add_variable = (GtkButton *)
02449         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02450     g_signal_connect (window->button_add_variable, "clicked",
02451         window_add_variable,
02452         NULL);
02453     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02454         _("Add variable"));
02455     window->button_remove_variable = (GtkButton *)
02456         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02457     g_signal_connect (window->button_remove_variable, "clicked",
02458         window_remove_variable, NULL);
02459     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02460         _("Remove variable"));
02461     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));

```

```

02449     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02450     gtk_widget_set_tooltip_text
02451         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02452     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02453     window->id_variable_label = g_signal_connect
02454         (window->entry_variable, "changed",
window_label_variable, NULL);
02455     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02456     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02457         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02458     gtk_widget_set_tooltip_text
02459         (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02460     window->scrolled_min
02461         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02462     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02463         GTK_WIDGET (window->spin_min));
02464     g_signal_connect (window->spin_min, "value-changed",
02465         window_rangemin_variable, NULL);
02466     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02467     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02468         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02469     gtk_widget_set_tooltip_text
02470         (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02471     window->scrolled_max
02472         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02473     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02474         GTK_WIDGET (window->spin_max));
02475     g_signal_connect (window->spin_max, "value-changed",
02476         window_rangemax_variable, NULL);
02477     window->check_minabs = (GtkCheckButton *)
02478         gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02479     g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
02480     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02481         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02482     gtk_widget_set_tooltip_text
02483         (GTK_WIDGET (window->spin_minabs),
02484         _("Minimum allowed value of the variable"));
02485     window->scrolled_minabs
02486         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02487     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02488         GTK_WIDGET (window->spin_minabs));
02489     g_signal_connect (window->spin_minabs, "value-changed",
02490         window_rangeminabs_variable, NULL);
02491     window->check_maxabs = (GtkCheckButton *)
02492         gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02493     g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02494     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02495         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02496     gtk_widget_set_tooltip_text
02497         (GTK_WIDGET (window->spin_maxabs),
02498         _("Maximum allowed value of the variable"));
02499     window->scrolled_maxabs
02500         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02501     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02502         GTK_WIDGET (window->spin_maxabs));
02503     g_signal_connect (window->spin_maxabs, "value-changed",
02504         window_rangemaxabs_variable, NULL);
02505     window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02506     window->spin_precision = (GtkSpinButton *)
02507         gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02508     gtk_widget_set_tooltip_text
02509         (GTK_WIDGET (window->spin_precision),
02510         _("Number of precision floating point digits\n"
02511         "0 is for integer numbers"));
02512     g_signal_connect (window->spin_precision, "value-changed",
02513         window_precision_variable, NULL);
02514     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02515     window->spin_sweeps =
02516         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02517     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02518         _("Number of steps sweeping the variable"));
02519     g_signal_connect (window->spin_sweeps, "value-changed",
02520         window_update_variable, NULL);
02521     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02522     window->spin_bits
02523         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02524     gtk_widget_set_tooltip_text
02525         (GTK_WIDGET (window->spin_bits),
02526         _("Number of bits to encode the variable"));
02527     g_signal_connect
02528         (window->spin_bits, "value-changed", window_update_variable, NULL);
;
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);
02558 gtk_grid_attach (window->grid_variable,
02559                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02560 gtk_grid_attach (window->grid_variable,
02561                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02562 gtk_grid_attach (window->grid_variable,
02563                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02564 gtk_grid_attach (window->grid_variable,
02565                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02566 gtk_grid_attach (window->grid_variable,
02567                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02568 gtk_grid_attach (window->grid_variable,
02569                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02570 gtk_grid_attach (window->grid_variable,
02571                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02572 gtk_grid_attach (window->grid_variable,
02573                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02574 gtk_grid_attach (window->grid_variable,
02575                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02576 gtk_grid_attach (window->grid_variable,
02577                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02578 gtk_grid_attach (window->grid_variable,
02579                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02580 gtk_grid_attach (window->grid_variable,
02581                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02582 gtk_grid_attach (window->grid_variable,
02583                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02584 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02585 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02586                   GTK_WIDGET (window->grid_variable));
02587
02588 // Creating the experiment widgets
02589 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02590 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02591                              _("Experiment selector"));
02592 window->id_experiment = g_signal_connect
02593     (window->combo_experiment, "changed",
02594      window_set_experiment, NULL);
02595 window->button_add_experiment = (GtkButton *)
02596     gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02597 g_signal_connect (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);

```



```

02616 gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02617 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02618 window->spin_weight
02619 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02620 gtk_widget_set_tooltip_text
02621 (GTK_WIDGET (window->spin_weight),
02622 _("Weight factor to build the objective function"));
02623 g_signal_connect
02624 (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
02625 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02626 gtk_grid_attach (window->grid_experiment,
02627 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02628 gtk_grid_attach (window->grid_experiment,
02629 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02630 gtk_grid_attach (window->grid_experiment,
02631 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
;
02632 gtk_grid_attach (window->grid_experiment,
02633 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02634 gtk_grid_attach (window->grid_experiment,
02635 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02636 gtk_grid_attach (window->grid_experiment,
02637 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02638 gtk_grid_attach (window->grid_experiment,
02639 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02640 for (i = 0; i < MAX_NINPITS; ++i)
02641 {
02642     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02643     window->check_template[i] = (GtkCheckButton *)
02644     gtk_check_button_new_with_label (buffer3);
02645     window->id_template[i]
02646     = g_signal_connect (window->check_template[i], "toggled",
02647     window_inputs_experiment, NULL);
02648     gtk_grid_attach (window->grid_experiment,
02649     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02650     window->button_template[i] = (GtkFileChooserButton *)
02651     gtk_file_chooser_button_new (_("Input template"),
02652     GTK_FILE_CHOOSER_ACTION_OPEN);
02653     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02654     _("Experimental input template file"));
02655     window->id_input[i] =
02656     g_signal_connect_swapped (window->button_template[i],
02657     "selection-changed",
02658     (GCallback) window_template_experiment,
02659     (void *) (size_t) i);
02660     gtk_grid_attach (window->grid_experiment,
02661     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02662 }
02663 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02664 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02665     GTK_WIDGET (window->grid_experiment));
02666
02667 // Creating the error norm widgets
02668 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02669 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02670 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02671     GTK_WIDGET (window->grid_norm));
02672 window->button_norm[0] = (GtkRadioButton *)
02673     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02674 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02675     tip_norm[0]);
02676 gtk_grid_attach (window->grid_norm,
02677     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02678 g_signal_connect (window->button_norm[0], "clicked",
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",
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_hexpand (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->
02703               scrolled_p),
02704                     1, 2, 1, 2);
02705     // Creating the grid and attaching the widgets to the grid
02706     window->grid = (GtkGrid *) gtk_grid_new ();
02707     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02708     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02709     gtk_grid_attach (window->grid,
02710                     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02711     gtk_grid_attach (window->grid,
02712                     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02713     gtk_grid_attach (window->grid,
02714                     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02715     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02716     gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
02717               window->grid));
02718     // Setting the window logo
02719     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02720     gtk_window_set_icon (window->window, window->logo);
02721     // Showing the window
02722     gtk_widget_show_all (GTK_WIDGET (window->window));
02723     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02724     #if GTK_MINOR_VERSION >= 16
02725     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02726     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02727     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02728     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02729     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02730     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02731     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02732     #endif
02733     // Reading initial example
02734     input_new ();
02735     buffer2 = g_get_current_dir ();
02736     buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02737     g_free (buffer2);
02738     window_read (buffer);
02739     g_free (buffer);
02740     #if DEBUG_INTERFACE
02741     fprintf (stderr, "window_new: start\n");
02742     #endif
02743 }

```

4.11.2.19 window_open()

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

Function to open the input data.

Definition at line 1980 of file [interface.c](#).

```

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

```



```

01995     dlg = (GtkFileChooserDialog *)
01996         gtk_file_chooser_dialog_new (_("Open input file"),
01997                                     window->window,
01998                                     GTK_FILE_CHOOSER_ACTION_OPEN,
01999                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02000                                     _("_OK"), GTK_RESPONSE_OK, NULL);
02001
02002     // Adding XML filter
02003     filter = (GtkFileFilter *) gtk_file_filter_new ();
02004     gtk_file_filter_set_name (filter, "XML");
02005     gtk_file_filter_add_pattern (filter, "*.xml");
02006     gtk_file_filter_add_pattern (filter, "*.XML");
02007     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02008
02009     // Adding JSON filter
02010     filter = (GtkFileFilter *) gtk_file_filter_new ();
02011     gtk_file_filter_set_name (filter, "JSON");
02012     gtk_file_filter_add_pattern (filter, "*.json");
02013     gtk_file_filter_add_pattern (filter, "*.JSON");
02014     gtk_file_filter_add_pattern (filter, "*.js");
02015     gtk_file_filter_add_pattern (filter, "*.JS");
02016     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02017
02018     // If OK saving
02019     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02020     {
02021
02022         // Traying to open the input file
02023         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02024         if (!window_read (buffer))
02025         {
02026             #if DEBUG_INTERFACE
02027                 fprintf (stderr, "window_open: error reading input file\n");
02028             #endif
02029             g_free (buffer);
02030
02031             // Reading backup file on error
02032             buffer = g_build_filename (directory, name, NULL);
02033             input->result = input->variables = NULL;
02034             if (!input_open (buffer))
02035             {
02036
02037                 // Closing on backup file reading error
02038                 #if DEBUG_INTERFACE
02039                     fprintf (stderr, "window_read: error reading backup file\n");
02040                 #endif
02041                 g_free (buffer);
02042                 break;
02043             }
02044             g_free (buffer);
02045         }
02046         else
02047         {
02048             g_free (buffer);
02049             break;
02050         }
02051     }
02052
02053     // Freeing and closing
02054     g_free (name);
02055     g_free (directory);
02056     gtk_widget_destroy (GTK_WIDGET (dlg));
02057     #if DEBUG_INTERFACE
02058         fprintf (stderr, "window_open: end\n");
02059     #endif
02060 }

```

4.11.2.20 window_precision_variable()

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

Function to update the variable precision in the main window.

Definition at line 1714 of file [interface.c](#).

```

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

```

4.11.2.21 window_rangemax_variable()

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

Function to update the variable rangemax in the main window.

Definition at line 1755 of file [interface.c](#).

```

01756 {
01757     unsigned int i;
01758     #if DEBUG_INTERFACE
01759         fprintf (stderr, "window_rangemax_variable: start\n");
01760     #endif
01761     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01762     input->variable[i].rangemax = gtk_spin_button_get_value (
    window->spin_max);
01763     #if DEBUG_INTERFACE
01764         fprintf (stderr, "window_rangemax_variable: end\n");
01765     #endif
01766 }

```

4.11.2.22 window_rangemaxabs_variable()

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

Function to update the variable rangemaxabs in the main window.

Definition at line 1790 of file [interface.c](#).

```

01791 {
01792     unsigned int i;
01793     #if DEBUG_INTERFACE
01794         fprintf (stderr, "window_rangemaxabs_variable: start\n");
01795     #endif
01796     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01797     input->variable[i].rangemaxabs
01798     = gtk_spin_button_get_value (window->spin_maxabs);
01799     #if DEBUG_INTERFACE
01800         fprintf (stderr, "window_rangemaxabs_variable: end\n");
01801     #endif
01802 }

```

4.11.2.23 window_rangemin_variable()

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

Function to update the variable rangemin in the main window.

Definition at line 1738 of file [interface.c](#).

```
01739 {
01740     unsigned int i;
01741     #if DEBUG_INTERFACE
01742     fprintf (stderr, "window_rangemin_variable: start\n");
01743     #endif
01744     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01745     input->variable[i].rangemin = gtk_spin_button_get_value (
01746         window->spin_min);
01747     #if DEBUG_INTERFACE
01748     fprintf (stderr, "window_rangemin_variable: end\n");
01749     #endif
01750 }
```

4.11.2.24 window_rangeminabs_variable()

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

Function to update the variable rangeminabs in the main window.

Definition at line 1772 of file [interface.c](#).

```
01773 {
01774     unsigned int i;
01775     #if DEBUG_INTERFACE
01776     fprintf (stderr, "window_rangeminabs_variable: start\n");
01777     #endif
01778     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01779     input->variable[i].rangeminabs
01780         = gtk_spin_button_get_value (window->spin_minabs);
01781     #if DEBUG_INTERFACE
01782     fprintf (stderr, "window_rangeminabs_variable: end\n");
01783     #endif
01784 }
```

4.11.2.25 window_read()

```
static int window_read (
    char * filename ) [static]
```

Function to read the input data of a file.

Returns

1 on succes, 0 on error.

Parameters

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

Definition at line 1864 of file [interface.c](#).

```

01865 {
01866     unsigned int i;
01867     char *buffer;
01868     #if DEBUG_INTERFACE
01869     fprintf (stderr, "window_read: start\n");
01870     #endif
01871
01872     // Reading new input file
01873     input_free ();
01874     input->result = input->variables = NULL;
01875     if (!input_open (filename))
01876     {
01877         #if DEBUG_INTERFACE
01878         fprintf (stderr, "window_read: end\n");
01879         #endif
01880         return 0;
01881     }
01882
01883     // Setting GTK+ widgets data
01884     gtk_entry_set_text (window->entry_result, input->result);
01885     gtk_entry_set_text (window->entry_variables, input->
variables);
01886     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01887     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01888     g_free (buffer);
01889     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
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
01897     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01901     switch (input->algorithm)
01902     {
01903         case ALGORITHM_MONTE_CARLO:
01904             gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
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
01915                 (GTK_TOGGLE_BUTTON (window->button_climbing[
input->climbing]),
TRUE);
01916                 gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01917                 gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
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             }
01925             break;
01926         default:

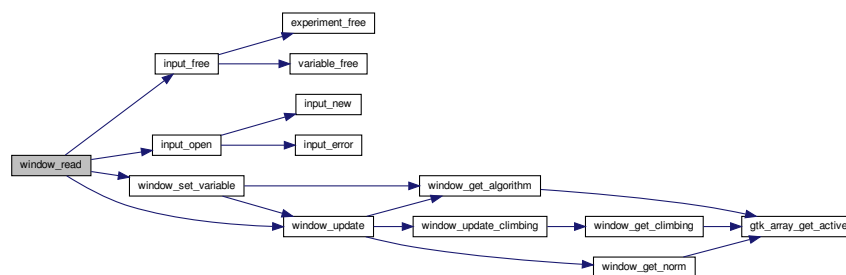
```

```

01933     gtk_spin_button_set_value (window->spin_population,
01934                               (gdouble) input->nsimulations);
01935     gtk_spin_button_set_value (window->spin_generations,
01936                               (gdouble) input->niterations);
01937     gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01938     gtk_spin_button_set_value (window->spin_reproduction,
01939                               input->reproduction_ratio);
01940     gtk_spin_button_set_value (window->spin_adaptation,
01941                               input->adaptation_ratio);
01942 }
01943 gtk_toggle_button_set_active
01944 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01945 gtk_spin_button_set_value (window->spin_p, input->p);
01946 gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01947 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01948 g_signal_handler_block (window->button_experiment,
01949                         window->id_experiment_name);
01950 gtk_combo_box_text_remove_all (window->combo_experiment);
01951 for (i = 0; i < input->nexperiments; ++i)
01952     gtk_combo_box_text_append_text (window->combo_experiment,
01953                                     input->experiment[i].name);
01954 g_signal_handler_unblock
01955 (window->button_experiment, window->
id_experiment_name);
01956 g_signal_handler_unblock (window->combo_experiment,
01957                         window->id_experiment);
01958 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01959 g_signal_handler_block (window->combo_variable, window->
id_variable);
01960 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01961 gtk_combo_box_text_remove_all (window->combo_variable);
01962 for (i = 0; i < input->nvariables; ++i)
01963     gtk_combo_box_text_append_text (window->combo_variable,
01964                                     input->variable[i].name);
01965 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01966 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01967 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01968 window_set_variable ();
01969 window_update ();
01970 #if DEBUG_INTERFACE
01971 fprintf (stderr, "window_read: end\n");
01972 #endif
01973 return 1;
01974 }

```

Here is the call graph for this function:



4.11.2.26 window_remove_experiment()

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

Function to remove an experiment in the main window.

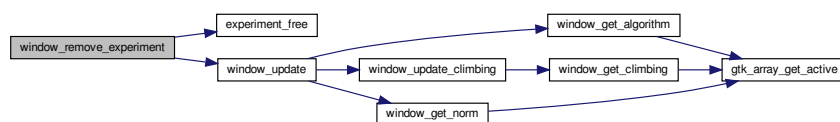
Definition at line 1356 of file [interface.c](#).

```

01357 {
01358     unsigned int i, j;
01359     #if DEBUG_INTERFACE
01360     fprintf (stderr, "window_remove_experiment: start\n");
01361     #endif
01362     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01363     g_signal_handler_block (window->combo_experiment, window->
01364         id_experiment);
01364     gtk_combo_box_text_remove (window->combo_experiment, i);
01365     g_signal_handler_unblock (window->combo_experiment,
01366         window->id_experiment);
01366     experiment_free (input->experiment + i, input->
01367         type);
01367     --input->nexperiments;
01368     for (j = i; j < input->nexperiments; ++j)
01369         memcpy (input->experiment + j, input->experiment + j + 1,
01370             sizeof (Experiment));
01371     j = input->nexperiments - 1;
01372     if (i > j)
01373         i = j;
01374     for (j = 0; j < input->experiment->ninputs; ++j)
01375         g_signal_handler_block (window->button_template[j],
01376             window->id_input[j]);
01376     g_signal_handler_block
01377         (window->button_experiment, window->
01378             id_experiment_name);
01378     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01379     g_signal_handler_unblock
01380         (window->button_experiment, window->
01381             id_experiment_name);
01381     for (j = 0; j < input->experiment->ninputs; ++j)
01382         g_signal_handler_unblock (window->button_template[j],
01383             window->id_input[j]);
01383     window_update ();
01384     #if DEBUG_INTERFACE
01385     fprintf (stderr, "window_remove_experiment: end\n");
01386     #endif
01387 }

```

Here is the call graph for this function:



4.11.2.27 window_remove_variable()

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

Function to remove a variable in the main window.

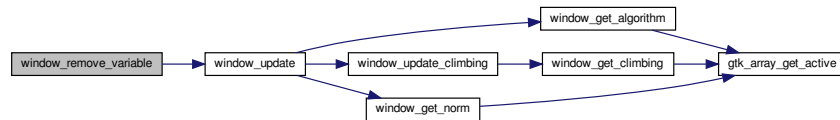
Definition at line 1626 of file [interface.c](#).

```

01627 {
01628     unsigned int i, j;
01629     #if DEBUG_INTERFACE
01630     fprintf (stderr, "window_remove_variable: start\n");
01631     #endif
01632     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01633     g_signal_handler_block (window->combo_variable, window->
id_variable);
01634     gtk_combo_box_text_remove (window->combo_variable, i);
01635     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01636     xmlFree (input->variable[i].name);
01637     --input->nvariables;
01638     for (j = i; j < input->nvariables; ++j)
01639         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01640     j = input->nvariables - 1;
01641     if (i > j)
01642         i = j;
01643     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01644     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01645     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01646     window_update ();
01647     #if DEBUG_INTERFACE
01648     fprintf (stderr, "window_remove_variable: end\n");
01649     #endif
01650 }

```

Here is the call graph for this function:



4.11.2.28 window_run()

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

Function to run a optimization.

Definition at line 975 of file [interface.c](#).

```

00976 {
00977     unsigned int i;
00978     char *msg, *msg2, buffer[64], buffer2[64];
00979     #if DEBUG_INTERFACE
00980     fprintf (stderr, "window_run: start\n");
00981     #endif
00982     if (!window_save ())
00983     {
00984         #if DEBUG_INTERFACE
00985             fprintf (stderr, "window_run: end\n");
00986         #endif
00987         return;
00988     }
00989     running_new ();
00990     while (gtk_events_pending ())
00991         gtk_main_iteration ();
00992     optimize_open ();
00993     #if DEBUG_INTERFACE
00994     fprintf (stderr, "window_run: closing running dialog\n");
00995     #endif

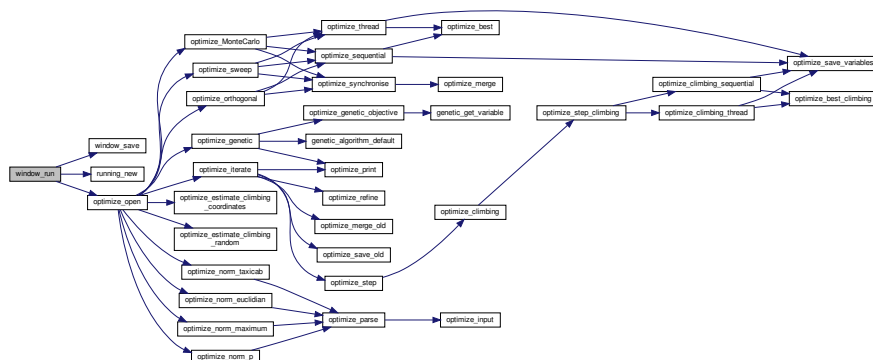
```

```

00996  gtk_spinner_stop (running->spinner);
00997  gtk_widget_destroy (GTK_WIDGET (running->dialog));
00998  #if DEBUG_INTERFACE
00999  fprintf (stderr, "window_run: displaying results\n");
01000  #endif
01001  snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01002  msg2 = g_strdup (buffer);
01003  for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01004  {
01005      snprintf (buffer, 64, "%s = %s\n",
01006               input->variable[i].name, format[input->
variable[i].precision]);
01007      snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008      msg = g_strconcat (msg2, buffer2, NULL);
01009      g_free (msg2);
01010  }
01011  snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01012           optimize->calculation_time);
01013  msg = g_strconcat (msg2, buffer, NULL);
01014  g_free (msg2);
01015  show_message (_("Best result"), msg, INFO_TYPE);
01016  g_free (msg);
01017  #if DEBUG_INTERFACE
01018  fprintf (stderr, "window_run: freeing memory\n");
01019  #endif
01020  optimize_free ();
01021  #if DEBUG_INTERFACE
01022  fprintf (stderr, "window_run: end\n");
01023  #endif
01024  }

```

Here is the call graph for this function:



4.11.2.29 window_save()

```
static int window_save ( ) [static]
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 824 of file [interface.c](#).


```

00825 {
00826     GtkFileChooserDialog *dlg;
00827     GtkFileFilter *filter1, *filter2;
00828     char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831     fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834     // Opening the saving dialog
00835     dlg = (GtkFileChooserDialog *)
00836         gtk_file_chooser_dialog_new (_("Save file"),
00837                                     window->window,
00838                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00840                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00841     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00842     buffer = g_build_filename (input->directory, input->name, NULL);
00843     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00844     g_free (buffer);
00845
00846     // Adding XML filter
00847     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00848     gtk_file_filter_set_name (filter1, "XML");
00849     gtk_file_filter_add_pattern (filter1, "*.xml");
00850     gtk_file_filter_add_pattern (filter1, "*.XML");
00851     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00852
00853     // Adding JSON filter
00854     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00855     gtk_file_filter_set_name (filter2, "JSON");
00856     gtk_file_filter_add_pattern (filter2, "*.json");
00857     gtk_file_filter_add_pattern (filter2, "*.JSON");
00858     gtk_file_filter_add_pattern (filter2, "*.js");
00859     gtk_file_filter_add_pattern (filter2, "*.JS");
00860     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00861
00862     if (input->type == INPUT_TYPE_XML)
00863         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00864     else
00865         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00866
00867     // If OK response then saving
00868     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00869     {
00870         // Setting input file type
00871         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00872         buffer = (char *) gtk_file_filter_get_name (filter1);
00873         if (!strcmp (buffer, "XML"))
00874             input->type = INPUT_TYPE_XML;
00875         else
00876             input->type = INPUT_TYPE_JSON;
00877
00878         // Adding properties to the root XML node
00879         input->simulator = gtk_file_chooser_get_filename
00880             (GTK_FILE_CHOOSER (window->button_simulator));
00881         if (gtk_toggle_button_get_active
00882             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00883             input->evaluator = gtk_file_chooser_get_filename
00884                 (GTK_FILE_CHOOSER (window->button_evaluator));
00885         else
00886             input->evaluator = NULL;
00887         if (input->type == INPUT_TYPE_XML)
00888         {
00889             input->result
00890                 = (char *) xmlStrdup ((const xmlChar *)
00891                                         gtk_entry_get_text (window->entry_result));
00892             input->variables
00893                 = (char *) xmlStrdup ((const xmlChar *)
00894                                         gtk_entry_get_text (window->
00895 entry_variables));
00896         }
00897         else
00898         {
00899             input->result = g_strdup (gtk_entry_get_text (window->
00900 entry_result));
00901             input->variables =
00902                 g_strdup (gtk_entry_get_text (window->entry_variables));
00903         }
00904
00905         // Setting the algorithm
00906         switch (window_get_algorithm ())
00907         {
00908             case ALGORITHM_MONTE_CARLO:
00909                 input->algorithm = ALGORITHM_MONTE_CARLO;
00910                 input->nsimulations
00911                     = gtk_spin_button_get_value_as_int (window->spin_simulations);

```

```

00910         input->niterations
00911         = gtk_spin_button_get_value_as_int (window->spin_iterations);
00912         input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00913         input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00914         window_save_climbing ();
00915         break;
00916         case ALGORITHM_SWEEP:
00917         input->algorithm = ALGORITHM_SWEEP;
00918         input->niterations
00919         = gtk_spin_button_get_value_as_int (window->spin_iterations);
00920         input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00921         input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00922         window_save_climbing ();
00923         break;
00924         case ALGORITHM_ORTHOGONAL:
00925         input->algorithm = ALGORITHM_ORTHOGONAL;
00926         input->niterations
00927         = gtk_spin_button_get_value_as_int (window->spin_iterations);
00928         input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00929         input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00930         window_save_climbing ();
00931         break;
00932         default:
00933         input->algorithm = ALGORITHM_GENETIC;
00934         input->nsimulations
00935         = gtk_spin_button_get_value_as_int (window->spin_population);
00936         input->niterations
00937         = gtk_spin_button_get_value_as_int (window->spin_generations);
00938         input->mutation_ratio
00939         = gtk_spin_button_get_value (window->spin_mutation);
00940         input->reproduction_ratio
00941         = gtk_spin_button_get_value (window->spin_reproduction);
00942         input->adaptation_ratio
00943         = gtk_spin_button_get_value (window->spin_adaptation);
00944         break;
00945     }
00946     input->norm = window_get_norm ();
00947     input->p = gtk_spin_button_get_value (window->spin_p);
00948     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00949
00950     // Saving the XML file
00951     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00952     input_save (buffer);
00953
00954     // Closing and freeing memory
00955     g_free (buffer);
00956     gtk_widget_destroy (GTK_WIDGET (dlg));
00957 #if DEBUG_INTERFACE
00958     fprintf (stderr, "window_save: end\n");
00959 #endif
00960     return 1;
00961 }
00962
00963 // Closing and freeing memory
00964 gtk_widget_destroy (GTK_WIDGET (dlg));
00965 #if DEBUG_INTERFACE
00966 fprintf (stderr, "window_save: end\n");
00967 #endif
00968 return 0;
00969 }

```

4.11.2.30 window_save_climbing()

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

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

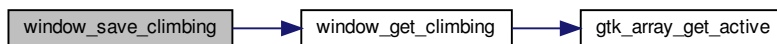
Definition at line 791 of file [interface.c](#).

```

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

```

Here is the call graph for this function:



4.11.2.31 window_set_algorithm()

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

Function to avoid memory errors changing the algorithm.

Definition at line 1282 of file [interface.c](#).

```

01283 {
01284     int i;
01285     #if DEBUG_INTERFACE
01286         fprintf (stderr, "window_set_algorithm: start\n");
01287     #endif
01288     i = window_get_algorithm ();
01289     switch (i)
01290     {
01291         case ALGORITHM_SWEEP:
01292         case ALGORITHM_ORTHOGONAL:
01293             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01294             if (i < 0)
01295                 i = 0;
01296             gtk_spin_button_set_value (window->spin_sweeps,
01297                                     (gdouble) input->variable[i].
01298                                     nsweeps);
01299             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;

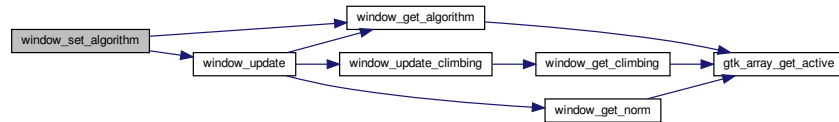
```

```

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()

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

Function to set the experiment data in the main window.

Definition at line 1316 of file [interface.c](#).

```

01317 {
01318     unsigned int i, j;
01319     char *buffer1, *buffer2;
01320     #if DEBUG_INTERFACE
01321     fprintf (stderr, "window_set_experiment: start\n");
01322     #endif
01323     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01324     gtk_spin_button_set_value (window->spin_weight, input->
01325                               experiment[i].weight);
01326     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01327     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01328     g_free (buffer1);
01329     g_signal_handler_block
01330         (window->button_experiment, window->
01331          id_experiment_name);
01332     gtk_file_chooser_set_filename
01333         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01334     g_signal_handler_unblock
01335         (window->button_experiment, window->
01336          id_experiment_name);
01337     g_free (buffer2);
01338     for (j = 0; j < input->experiment->ninputs; ++j)
01339     {
01340         g_signal_handler_block (window->button_template[j],
01341                                window->id_input[j]);
01342         buffer2 =
01343             g_build_filename (input->directory, input->experiment[i].
01344                             stencil[j],
01345                             NULL);
01346         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01347                                     (window->button_template[j]), buffer2);
01348         g_free (buffer2);
01349         g_signal_handler_unblock
01350             (window->button_template[j], window->id_input[j]);
01351     }
01352     #if DEBUG_INTERFACE
01353     fprintf (stderr, "window_set_experiment: end\n");
01354     #endif
01355 }

```

4.11.2.33 window_set_variable()

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

Function to set the variable data in the main window.

Definition at line 1549 of file [interface.c](#).

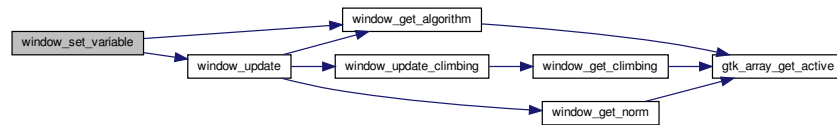
```
01550 {
01551     unsigned int i;
01552     #if DEBUG_INTERFACE
01553         fprintf (stderr, "window_set_variable: start\n");
01554     #endif
01555     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01556     g_signal_handler_block (window->entry_variable, window->
01557         id_variable_label);
01557     gtk_entry_set_text (window->entry_variable, input->
01558         variable[i].name);
01558     g_signal_handler_unblock (window->entry_variable, window->
01559         id_variable_label);
01559     gtk_spin_button_set_value (window->spin_min, input->
01560         variable[i].rangemin);
01560     gtk_spin_button_set_value (window->spin_max, input->
01561         variable[i].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     {
01583         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01584         gtk_toggle_button_set_active
01585             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01586     }
01587     gtk_spin_button_set_value (window->spin_precision,
01588         input->variable[i].precision);
01589     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01590         nsteps);
01590     if (input->nsteps)
01591         gtk_spin_button_set_value (window->spin_step, input->
01592             variable[i].step);
01592     #if DEBUG_INTERFACE
01593         fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01594             input->variable[i].precision);
01595     #endif
01596     switch (window_get_algorithm ())
01597     {
01598     case ALGORITHM_SWEEP:
01599     case ALGORITHM_ORTHOGONAL:
01600         gtk_spin_button_set_value (window->spin_sweeps,
01601             (gdouble) input->variable[i].
01602             nsweeps);
01602     #if DEBUG_INTERFACE
01603         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01604             input->variable[i].nsweeps);
01605     #endif
01606         break;
01607     case ALGORITHM_GENETIC:
01608         gtk_spin_button_set_value (window->spin_bits,
01609             (gdouble) input->variable[i].nbits);
01610     #if DEBUG_INTERFACE
01611         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01612             input->variable[i].nbits);
01613     #endif
01614         break;
01615     }
```

```

01616     window_update ();
01617     #if DEBUG_INTERFACE
01618     fprintf (stderr, "window_set_variable: end\n");
01619     #endif
01620 }

```

Here is the call graph for this function:



4.11.2.34 window_step_variable()

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

Function to update the variable step in the main window.

Definition at line 1808 of file [interface.c](#).

```

01809 {
01810     unsigned int i;
01811     #if DEBUG_INTERFACE
01812     fprintf (stderr, "window_step_variable: start\n");
01813     #endif
01814     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01815     input->variable[i].step = gtk_spin_button_get_value (window->
        spin_step);
01816     #if DEBUG_INTERFACE
01817     fprintf (stderr, "window_step_variable: end\n");
01818     #endif
01819 }

```

4.11.2.35 window_template_experiment()

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

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

Parameters

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

Definition at line 1518 of file [interface.c](#).

```

01520 {
01521     unsigned int i, j;
01522     char *buffer;
01523     GFile *file1, *file2;
01524     #if DEBUG_INTERFACE
01525     fprintf (stderr, "window_template_experiment: start\n");
01526     #endif
01527     i = (size_t) data;
01528     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01529     file1
01530     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01531     file2 = g_file_new_for_path (input->directory);
01532     buffer = g_file_get_relative_path (file2, file1);
01533     if (input->type == INPUT_TYPE_XML)
01534         input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01535     else
01536         input->experiment[j].stencil[i] = g_strdup (buffer);
01537     g_free (buffer);
01538     g_object_unref (file2);
01539     g_object_unref (file1);
01540     #if DEBUG_INTERFACE
01541     fprintf (stderr, "window_template_experiment: end\n");
01542     #endif
01543 }

```

4.11.2.36 window_update()

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

Function to update the main window view.

Definition at line 1125 of file [interface.c](#).

```

01126 {
01127     unsigned int i;
01128     #if DEBUG_INTERFACE
01129     fprintf (stderr, "window_update: start\n");
01130     #endif
01131     gtk_widget_set_sensitive
01132     (GTK_WIDGET (window->button_evaluator),
01133      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01134      (window->check_evaluator)));
01135     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01136     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01137     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01138     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01139     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01140     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01141     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01142     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01143     gtk_widget_hide (GTK_WIDGET (window->label_population));
01144     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01145     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01146     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01147     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01148     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01149     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01150     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01151     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01152     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01153     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01154     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01155     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01156     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01157     gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01158     gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01159     gtk_widget_hide (GTK_WIDGET (window->label_step));
01160     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01161     gtk_widget_hide (GTK_WIDGET (window->label_p));
01162     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01163     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01164     switch (window_get_algorithm ())
01165     {
01166     case ALGORITHM_MONTE_CARLO:
01167         gtk_widget_show (GTK_WIDGET (window->label_simulations));

```

```

01168     gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01169     gtk_widget_show (GTK_WIDGET (window->label_iterations));
01170     gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01171     if (i > 1)
01172     {
01173         gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01174         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01175         gtk_widget_show (GTK_WIDGET (window->label_bests));
01176         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01177     }
01178     window_update_climbing ();
01179     break;
01180 case ALGORITHM_SWEEP:
01181 case ALGORITHM_ORTHOGONAL:
01182     gtk_widget_show (GTK_WIDGET (window->label_iterations));
01183     gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01184     if (i > 1)
01185     {
01186         gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01187         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01188         gtk_widget_show (GTK_WIDGET (window->label_bests));
01189         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01190     }
01191     gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01192     gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01193     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01194     window_update_climbing ();
01195     break;
01196 default:
01197     gtk_widget_show (GTK_WIDGET (window->label_population));
01198     gtk_widget_show (GTK_WIDGET (window->spin_population));
01199     gtk_widget_show (GTK_WIDGET (window->label_generations));
01200     gtk_widget_show (GTK_WIDGET (window->spin_generations));
01201     gtk_widget_show (GTK_WIDGET (window->label_mutation));
01202     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01203     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01204     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01205     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01206     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01207     gtk_widget_show (GTK_WIDGET (window->label_bits));
01208     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01209 }
01210 gtk_widget_set_sensitive
01211 (GTK_WIDGET (window->button_remove_experiment),
01212  input->nexperiments > 1);
01212 gtk_widget_set_sensitive
01213 (GTK_WIDGET (window->button_remove_variable),
01214  input->nvariables > 1);
01214 for (i = 0; i < input->experiment->ninputs; ++i)
01215 {
01216     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01217     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01218     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01219     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01220     g_signal_handler_block
01221     (window->check_template[i], window->
01222      id_template[i]);
01222     g_signal_handler_block (window->button_template[i],
01223      window->id_input[i]);
01223     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01224      (window->check_template[i]), 1);
01225     g_signal_handler_unblock (window->button_template[i],
01226      window->id_input[i]);
01227     g_signal_handler_unblock (window->check_template[i],
01228      window->id_template[i]);
01229 }
01230 if (i > 0)
01231 {
01232     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01233     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01234      gtk_toggle_button_get_active
01235      GTK_TOGGLE_BUTTON (window->check_template
01236      [i - 1]));
01237 }
01238 if (i < MAX_NINPUTS)
01239 {
01240     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01241     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01242     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01243     gtk_widget_set_sensitive
01244     (GTK_WIDGET (window->button_template[i]),
01245      gtk_toggle_button_get_active
01246      GTK_TOGGLE_BUTTON (window->check_template[i]));
01247     g_signal_handler_block
01248     (window->check_template[i], window->
01249      id_template[i]);
01249     g_signal_handler_block (window->button_template[i],

```

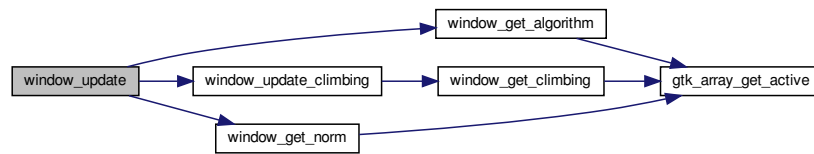


```

    window->id_input[i]);
01250     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01251                                   (window->check_template[i]), 0);
01252     g_signal_handler_unblock (window->button_template[i],
01253                               window->id_input[i]);
01254     g_signal_handler_unblock (window->check_template[i],
01255                               window->id_template[i]);
01256 }
01257 while (++i < MAX_NINPUTS)
01258 {
01259     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01260     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01261 }
01262 gtk_widget_set_sensitive
01263 (GTK_WIDGET (window->spin_minabs),
01264  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01265 gtk_widget_set_sensitive
01266 (GTK_WIDGET (window->spin_maxabs),
01267  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01268 if (window_get_norm () == ERROR_NORM_P)
01269 {
01270     gtk_widget_show (GTK_WIDGET (window->label_p));
01271     gtk_widget_show (GTK_WIDGET (window->spin_p));
01272 }
01273 #if DEBUG_INTERFACE
01274 fprintf (stderr, "window_update: end\n");
01275 #endif
01276 }

```

Here is the call graph for this function:



4.11.2.37 window_update_climbing()

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

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

Definition at line 1094 of file [interface.c](#).

```

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

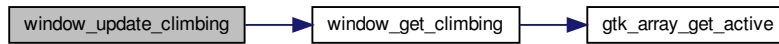
```

```

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()

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

Function to update the variable data in the main window.

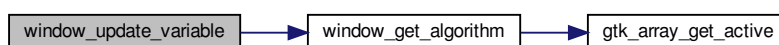
Definition at line 1825 of file [interface.c](#).

```

01826 {
01827     int i;
01828 #if DEBUG_INTERFACE
01829     fprintf (stderr, "window_update_variable: start\n");
01830 #endif
01831     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01832     if (i < 0)
01833         i = 0;
01834     switch (window_get_algorithm ())
01835     {
01836     case ALGORITHM_SWEEP:
01837     case ALGORITHM_ORTHOGONAL:
01838         input->variable[i].nsweeps
01839         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01840 #if DEBUG_INTERFACE
01841         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01842                 input->variable[i].nsweeps);
01843 #endif
01844         break;
01845     case ALGORITHM_GENETIC:
01846         input->variable[i].nbits
01847         = gtk_spin_button_get_value_as_int (window->spin_bits);
01848 #if DEBUG_INTERFACE
01849         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01850                 input->variable[i].nbits);
01851 #endif
01852     }
01853 #if DEBUG_INTERFACE
01854     fprintf (stderr, "window_update_variable: end\n");
01855 #endif
01856 }

```

Here is the call graph for this function:



4.11.2.39 window_weight_experiment()

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

Function to update the experiment weight in the main window.

Definition at line 1476 of file [interface.c](#).

```
01477 {
01478     unsigned int i;
01479     #if DEBUG_INTERFACE
01480     fprintf (stderr, "window_weight_experiment: start\n");
01481     #endif
01482     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01483     input->experiment[i].weight = gtk_spin_button_get_value (
01484         window->spin_weight);
01485     #if DEBUG_INTERFACE
01486     fprintf (stderr, "window_weight_experiment: end\n");
01487     #endif
01488 }
```

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"
```

```

00060 #include "experiment.h"
00061 #include "variable.h"
00062 #include "input.h"
00063 #include "optimize.h"
00064 #include "interface.h"
00065
00066 #define DEBUG_INTERFACE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define INPUT_FILE "test-ga-win.xml"
00074 #else
00075 #define INPUT_FILE "test-ga.xml"
00076 #endif
00077
00078 Window window[1];
00080
00081 static const char *logo[] = {
00082     "32 32 3 1",
00083     "    c None",
00084     ".    c #0000FF",
00085     "+    c #FF0000",
00086     " ",
00087     " ",
00088     " ",
00089     " . . . .",
00090     " . . . .",
00091     " . . . .",
00092     " . . . .",
00093     " . . . .",
00094     " . . . .",
00095     " . . . .",
00096     " . . . .",
00097     " . . . .",
00098     " . . . .",
00099     " . . . .",
00100     " . . . .",
00101     " . . . .",
00102     " . . . .",
00103     " . . . .",
00104     " . . . .",
00105     " . . . .",
00106     " . . . .",
00107     " . . . .",
00108     " . . . .",
00109     " . . . .",
00110     " . . . .",
00111     " . . . .",
00112     " . . . .",
00113     " . . . .",
00114     " . . . .",
00115     " . . . .",
00116     " . . . .",
00117     " . . . .",
00118 };
00119
00120 /*
00121 const char * logo[] = {
00122     "32 32 3 1",
00123     "    c #FFFFFFFF",
00124     ".    c #00000000FFFF",
00125     "X    c #FFFF00000000",
00126     " ",
00127     " ",
00128     " ",
00129     " . . . .",
00130     " . . . .",
00131     " . . . .",
00132     " . . . .",
00133     " . . . .",
00134     " . . . .",
00135     " . . . .",
00136     " . . . .",
00137     " . . . .",
00138     " . . . .",
00139     " . . . .",
00140     " . . . .",
00141     " . . . .",
00142     " . . . .",
00143     " . . . .",
00144     " . . . .",
00145     " . . . .",
00146     " . . . .",
00147     " . . . .",
00148     " . . . .",
00149     " . . . .",
00150     " . . . .",

```

```

00151 " . . . . . "
00152 " . . . . . "
00153 " . . . . . "
00154 " . . . . . "
00155 " . . . . . "
00156 " . . . . . "
00157 " . . . . . ";
00158 */
00159
00160 static Options options[1];
00162 static Running running[1];
00164
00168 static void
00169 input_save_climbing_xml (xmlNode * node)
00170 {
00171     #if DEBUG_INTERFACE
00172         fprintf (stderr, "input_save_climbing_xml: start\n");
00173     #endif
00174     if (input->nsteps)
00175     {
00176         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
input->nsteps);
00177         if (input->relaxation != DEFAULT_RELAXATION)
00178             xml_node_set_float (node, (const xmlChar *)
LABEL_RELAXATION,
input->relaxation);
00179         switch (input->climbing)
00180         {
00181             case CLIMBING_METHOD_COORDINATES:
00182                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
(const xmlChar *) LABEL_COORDINATES);
00183                 break;
00184             default:
00185                 xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
(const xmlChar *) LABEL_RANDOM);
00186                 xml_node_set_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
input->nestimates);
00187             }
00188         }
00189     #if DEBUG_INTERFACE
00190         fprintf (stderr, "input_save_climbing_xml: end\n");
00191     #endif
00192 }
00193
00201 static void
00202 input_save_climbing_json (JsonNode * node)
00203 {
00204     JsonObject *object;
00205     #if DEBUG_INTERFACE
00206         fprintf (stderr, "input_save_climbing_json: start\n");
00207     #endif
00208     object = json_node_get_object (node);
00209     if (input->nsteps)
00210     {
00211         json_object_set_uint (object, LABEL_NSTEPS,
input->nsteps);
00212         if (input->relaxation != DEFAULT_RELAXATION)
00213             json_object_set_float (object, LABEL_RELAXATION,
input->relaxation);
00214         switch (input->climbing)
00215         {
00216             case CLIMBING_METHOD_COORDINATES:
00217                 json_object_set_string_member (object, LABEL_CLIMBING,
LABEL_COORDINATES);
00218                 break;
00219             default:
00220                 json_object_set_string_member (object, LABEL_CLIMBING,
LABEL_RANDOM);
00221                 json_object_set_uint (object, LABEL_NESTIMATES,
input->nestimates);
00222             }
00223         }
00224     #if DEBUG_INTERFACE
00225         fprintf (stderr, "input_save_climbing_json: end\n");
00226     #endif
00227 }
00228
00233 static inline void
00234 input_save_xml (xmlDoc * doc)
00235 {
00236     unsigned int i, j;
00237     char *buffer;
00238     xmlNode *node, *child;
00239     GFile *file, *file2;
00240
00241     #if DEBUG_INTERFACE

```

```

00242     fprintf (stderr, "input_save_xml: start\n");
00243 #endif
00244
00245 // Setting root XML node
00246 node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00247 xmlDocSetRootElement (doc, node);
00248
00249 // Adding properties to the root XML node
00250 if (xmlStrcmp
00251     ((const xmlChar *) input->result, (const xmlChar *) result_name))
00252     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00253                 (xmlChar *) input->result);
00254 if (xmlStrcmp
00255     ((const xmlChar *) input->variables, (const xmlChar *)
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     }
00274 if (input->seed != DEFAULT_RANDOM_SEED)
00275     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);
00276
00277 // Setting the algorithm
00278 buffer = (char *) g_slice_alloc (64);
00279 switch (input->algorithm)
00280     {
00281     case ALGORITHM_MONTE_CARLO:
00282         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00283                     (const xmlChar *) LABEL_MONTE_CARLO);
00284         snprintf (buffer, 64, "%u", input->nsimulations);
00285         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00286                     (xmlChar *) buffer);
00287         snprintf (buffer, 64, "%u", input->niterations);
00288         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00289                     (xmlChar *) buffer);
00290         snprintf (buffer, 64, "%.3lg", input->tolerance);
00291         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00292         snprintf (buffer, 64, "%u", input->nbest);
00293         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00294         input_save_climbing_xml (node);
00295         break;
00296     case ALGORITHM_SWEEP:
00297         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00298                     (const xmlChar *) LABEL_SWEEP);
00299         snprintf (buffer, 64, "%u", input->niterations);
00300         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00301                     (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%.3lg", input->tolerance);
00303         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00304         snprintf (buffer, 64, "%u", input->nbest);
00305         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00306         input_save_climbing_xml (node);
00307         break;
00308     case ALGORITHM_ORTHOGONAL:
00309         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00310                     (const xmlChar *) LABEL_ORTHOGONAL);
00311         snprintf (buffer, 64, "%u", input->niterations);
00312         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00313                     (xmlChar *) buffer);
00314         snprintf (buffer, 64, "%.3lg", input->tolerance);
00315         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00316         snprintf (buffer, 64, "%u", input->nbest);
00317         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00318         input_save_climbing_xml (node);
00319         break;
00320     default:
00321         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00322                     (const xmlChar *) LABEL_GENETIC);
00323         snprintf (buffer, 64, "%u", input->nsimulations);
00324         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00325                     (xmlChar *) buffer);
00326         snprintf (buffer, 64, "%u", input->niterations);

```

```

00327     xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00328                 (xmlChar *) buffer);
00329     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00330     xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00331     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00332     xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00333                 (xmlChar *) buffer);
00334     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00335     xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00336     break;
00337 }
00338 g_slice_free1 (64, buffer);
00339 if (input->threshold != 0.)
00340     xml_node_set_float (node, (const xmlChar *)
00341 LABEL_THRESHOLD,
00342                       input->threshold);
00343 // Setting the experimental data
00344 for (i = 0; i < input->nexperiments; ++i)
00345 {
00346     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00347     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00348                 (xmlChar *) input->experiment[i].name);
00349     if (input->experiment[i].weight != 1.)
00350         xml_node_set_float (child, (const xmlChar *)
00351 LABEL_WEIGHT,
00352                             input->experiment[i].weight);
00353     for (j = 0; j < input->experiment->ninputs; ++j)
00354         xmlSetProp (child, (const xmlChar *) stencil[j],
00355                     (xmlChar *) input->experiment[i].stencil[j]);
00356 }
00357 // Setting the variables data
00358 for (i = 0; i < input->nvariables; ++i)
00359 {
00360     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00361     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00362                 (xmlChar *) input->variable[i].name);
00363     xml_node_set_float (child, (const xmlChar *)
00364 LABEL_MINIMUM,
00365                         input->variable[i].rangemin);
00366     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00367         xml_node_set_float (child, (const xmlChar *)
00368 LABEL_ABSOLUTE_MINIMUM,
00369                             input->variable[i].rangeminabs);
00370     xml_node_set_float (child, (const xmlChar *)
00371 LABEL_MAXIMUM,
00372                         input->variable[i].rangemax);
00373     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00374         xml_node_set_float (child, (const xmlChar *)
00375 LABEL_ABSOLUTE_MAXIMUM,
00376                             input->variable[i].rangemaxabs);
00377     if (input->variable[i].precision !=
00378         DEFAULT_PRECISION)
00379         xml_node_set_uint (child, (const xmlChar *)
00380 LABEL_PRECISION,
00381                             input->variable[i].precision);
00382     if (input->algorithm == ALGORITHM_SWEEP
00383         || input->algorithm == ALGORITHM_ORTHOGONAL)
00384         xml_node_set_uint (child, (const xmlChar *)
00385 LABEL_NSWEEPS,
00386                             input->variable[i].nsweeps);
00387     else if (input->algorithm == ALGORITHM_GENETIC)
00388         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00389                             input->variable[i].nbits);
00390     if (input->nsteps)
00391         xml_node_set_float (child, (const xmlChar *)
00392 LABEL_STEP,
00393                             input->variable[i].step);
00394 }
00395 // Saving the error norm
00396 switch (input->norm)
00397 {
00398     case ERROR_NORM_MAXIMUM:
00399         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00400                     (const xmlChar *) LABEL_MAXIMUM);
00401         break;
00402     case ERROR_NORM_P:
00403         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00404                     (const xmlChar *) LABEL_P);
00405         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00406                             input->p);
00407         break;
00408     case ERROR_NORM_TAXICAB:
00409         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00410                     (const xmlChar *) LABEL_TAXICAB);

```

```

00403     }
00404
00405 #if DEBUG_INTERFACE
00406     fprintf (stderr, "input_save: end\n");
00407 #endif
00408 }
00409
00410 static inline void
00411 input_save_json (JsonGenerator * generator)
00412 {
00413     unsigned int i, j;
00414     char *buffer;
00415     JsonNode *node, *child;
00416     JsonObject *object;
00417     JsonArray *array;
00418     GFile *file, *file2;
00419
00420 #if DEBUG_INTERFACE
00421     fprintf (stderr, "input_save_json: start\n");
00422 #endif
00423
00424     // Setting root JSON node
00425     node = json_node_new (JSON_NODE_OBJECT);
00426     object = json_node_get_object (node);
00427     json_generator_set_root (generator, node);
00428
00429     // Adding properties to the root JSON node
00430     if (strcmp (input->result, result_name))
00431         json_object_set_string_member (object, LABEL_RESULT_FILE,
00432                                     input->result);
00433     if (strcmp (input->variables, variables_name))
00434         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00435                                     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,
00453                             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);

```



```

00489     snprintf (buffer, 64, "%u", input->nbest);
00490     json_object_set_string_member (object, LABEL_NBEST, buffer);
00491     input_save_climbing_json (node);
00492     break;
00493     default:
00494         json_object_set_string_member (object, LABEL_ALGORITHM,
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,
input->threshold);
00510
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,
input->experiment[i].name);
00518         if (input->experiment[i].weight != 1.)
00519             json_object_set_float (object, LABEL_WEIGHT,
input->experiment[i].weight);
00520         for (j = 0; j < input->experiment->ninputs; ++j)
00521             json_object_set_string_member (object, stencil[j],
input->experiment[i].
stencil[j]);
00522         json_array_add_element (array, child);
00523     }
00524     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00525
00526     // Setting the variables data
00527     array = json_array_new ();
00528     for (i = 0; i < input->nvariables; ++i)
00529     {
00530         child = json_node_new (JSON_NODE_OBJECT);
00531         object = json_node_get_object (child);
00532         json_object_set_string_member (object, LABEL_NAME,
input->variable[i].name);
00533         json_object_set_float (object, LABEL_MINIMUM,
input->variable[i].rangemin);
00534         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00535             json_object_set_float (object,
LABEL_ABSOLUTE_MINIMUM,
input->variable[i].rangeminabs);
00536         json_object_set_float (object, LABEL_MAXIMUM,
input->variable[i].rangemax);
00537         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00538             json_object_set_float (object,
LABEL_ABSOLUTE_MAXIMUM,
input->variable[i].rangemaxabs);
00539         if (input->variable[i].precision !=
DEFAULT_PRECISION)
00540             json_object_set_uint (object, LABEL_PRECISION,
input->variable[i].precision);
00541         if (input->algorithm == ALGORITHM_SWEEP
|| input->algorithm == ALGORITHM_ORTHOGONAL)
00542             json_object_set_uint (object, LABEL_NSWEEPS,
input->variable[i].nsweeps);
00543         else if (input->algorithm == ALGORITHM_GENETIC)
00544             json_object_set_uint (object, LABEL_NBITS,
input->variable[i].nbits);
00545         if (input->nsteps)
00546             json_object_set_float (object, LABEL_STEP,
input->variable[i].step);
00547         json_array_add_element (array, child);
00548     }
00549     json_object_set_array_member (object, LABEL_VARIABLES, array);
00550
00551     // Saving the error norm
00552     switch (input->norm)
00553     {
00554     case ERROR_NORM_MAXIMUM:
00555         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00556         break;

```

```

00568     case ERROR_NORM_P:
00569         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00570         json_object_set_float (object, LABEL_P, input->
p);
00571         break;
00572     case ERROR_NORM_TAXICAB:
00573         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00574     }
00575
00576 #if DEBUG_INTERFACE
00577     fprintf (stderr, "input_save_json: end\n");
00578 #endif
00579 }
00580
00584 static inline void
00585 input_save (char *filename)
00586 {
00587     xmlDoc *doc;
00588     JsonGenerator *generator;
00589
00590 #if DEBUG_INTERFACE
00591     fprintf (stderr, "input_save: start\n");
00592 #endif
00593
00594     // Getting the input file directory
00595     input->name = g_path_get_basename (filename);
00596     input->directory = g_path_get_dirname (filename);
00597
00598     if (input->type == INPUT_TYPE_XML)
00599     {
00600         // Opening the input file
00601         doc = xmlNewDoc ((const xmlChar *) "1.0");
00602         input_save_xml (doc);
00603
00604         // Saving the XML file
00605         xmlSaveFormatFile (filename, doc, 1);
00606
00607         // Freeing memory
00608         xmlFreeDoc (doc);
00609     }
00610     else
00611     {
00612         // Opening the input file
00613         generator = json_generator_new ();
00614         json_generator_set_pretty (generator, TRUE);
00615         input_save_json (generator);
00616
00617         // Saving the JSON file
00618         json_generator_to_file (generator, filename, NULL);
00619
00620         // Freeing memory
00621         g_object_unref (generator);
00622     }
00623
00624 #if DEBUG_INTERFACE
00625     fprintf (stderr, "input_save: end\n");
00626 #endif
00627 }
00628
00632 static void
00633 options_new ()
00634 {
00635     #if DEBUG_INTERFACE
00636         fprintf (stderr, "options_new: start\n");
00637     #endif
00638     options->label_seed = (GtkLabel *)
00639         gtk_label_new (_("Pseudo-random numbers generator seed"));
00640     options->spin_seed = (GtkSpinButton *)
00641         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00642     gtk_widget_set_tooltip_text
00643         (GTK_WIDGET (options->spin_seed),
00644         _("Seed to init the pseudo-random numbers generator"));
00645     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00646     options->label_threads = (GtkLabel *)
00647         gtk_label_new (_("Threads number for the stochastic algorithm"));
00648     options->spin_threads
00649         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00650     gtk_widget_set_tooltip_text
00651         (GTK_WIDGET (options->spin_threads),
00652         _("Number of threads to perform the calibration/optimization for "
00653         "the stochastic algorithm"));
00654     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00655     options->label_climbing = (GtkLabel *)
00656         gtk_label_new (_("Threads number for the hill climbing method"));
00657     options->spin_climbing =

```

```

00658     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00659     gtk_widget_set_tooltip_text
00660     (GTK_WIDGET (options->spin_climbing),
00661      _("Number of threads to perform the calibration/optimization for the "
00662       "hill climbing method"));
00663     gtk_spin_button_set_value (options->spin_climbing,
00664                               (gdouble) nthreads_climbing);
00665     options->grid = (GtkGrid *) gtk_grid_new ();
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00667     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00668     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00669                     0, 1, 1, 1);
00670     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00671                     1, 1, 1, 1);
00672     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
00673                     1);
00674     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
00675                     1);
00676     gtk_widget_show_all (GTK_WIDGET (options->grid));
00677     options->dialog = (GtkDialog *)
00678     gtk_dialog_new_with_buttons (_("Options"),
00679                                window->window,
00680                                GTK_DIALOG_MODAL,
00681                                _("_OK"), GTK_RESPONSE_OK,
00682                                _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00683     gtk_container_add
00684     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00685      GTK_WIDGET (options->grid));
00686     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00687     {
00688         input->seed
00689         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00690         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00691         nthreads_climbing
00692         = gtk_spin_button_get_value_as_int (options->spin_climbing);
00693     }
00694     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00695     #if DEBUG_INTERFACE
00696     fprintf (stderr, "options_new: end\n");
00697     #endif
00698 }
00699
00700 static inline void
00701 running_new ()
00702 {
00703     #if DEBUG_INTERFACE
00704     fprintf (stderr, "running_new: start\n");
00705     #endif
00706     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00707     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00708     running->grid = (GtkGrid *) gtk_grid_new ();
00709     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00710     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00711     running->dialog = (GtkDialog *)
00712     gtk_dialog_new_with_buttons (_("Calculating"),
00713                                window->window, GTK_DIALOG_MODAL, NULL, NULL);
00714     gtk_container_add (GTK_CONTAINER
00715                       (gtk_dialog_get_content_area (running->dialog)),
00716                        GTK_WIDGET (running->grid));
00717     gtk_spinner_start (running->spinner);
00718     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00719     #if DEBUG_INTERFACE
00720     fprintf (stderr, "running_new: end\n");
00721     #endif
00722 }
00723
00724 static unsigned int
00725 window_get_algorithm ()
00726 {
00727     unsigned int i;
00728     #if DEBUG_INTERFACE
00729     fprintf (stderr, "window_get_algorithm: start\n");
00730     #endif
00731     i = gtk_array_get_active (window->button_algorithm,
00732                              NALGORITHMS);
00733     #if DEBUG_INTERFACE
00734     fprintf (stderr, "window_get_algorithm: %u\n", i);
00735     fprintf (stderr, "window_get_algorithm: end\n");
00736     #endif
00737     return i;
00738 }
00739
00740 static unsigned int
00741 window_get_climbing ()
00742 {
00743     unsigned int i;
00744     #if DEBUG_INTERFACE

```

```

00757     fprintf (stderr, "window_get_climbing: start\n");
00758 #endif
00759     i = gtk_array_get_active (window->button_climbing,
00760                             NCLIMBINGS);
00760 #if DEBUG_INTERFACE
00761     fprintf (stderr, "window_get_climbing: %u\n", i);
00762     fprintf (stderr, "window_get_climbing: end\n");
00763 #endif
00764     return i;
00765 }
00766
00772 static unsigned int
00773 window_get_norm ()
00774 {
00775     unsigned int i;
00776 #if DEBUG_INTERFACE
00777     fprintf (stderr, "window_get_norm: start\n");
00778 #endif
00779     i = gtk_array_get_active (window->button_norm,
00780                             NNORMS);
00780 #if DEBUG_INTERFACE
00781     fprintf (stderr, "window_get_norm: %u\n", i);
00782     fprintf (stderr, "window_get_norm: end\n");
00783 #endif
00784     return i;
00785 }
00786
00790 static void
00791 window_save_climbing ()
00792 {
00793 #if DEBUG_INTERFACE
00794     fprintf (stderr, "window_save_climbing: start\n");
00795 #endif
00796     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_climbing)))
00797     {
00798         input->nsteps = gtk_spin_button_get_value_as_int (window->
00799 spin_steps);
00799         input->relaxation = gtk_spin_button_get_value (window->
00800 spin_relaxation);
00800         switch (window_get_climbing ())
00801         {
00802             case CLIMBING_METHOD_COORDINATES:
00803                 input->climbing = CLIMBING_METHOD_COORDINATES;
00804                 break;
00805             default:
00806                 input->climbing = CLIMBING_METHOD_RANDOM;
00807                 input->nestimates
00808                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00809         }
00810     }
00811     else
00812         input->nsteps = 0;
00813 #if DEBUG_INTERFACE
00814     fprintf (stderr, "window_save_climbing: end\n");
00815 #endif
00816 }
00817
00823 static int
00824 window_save ()
00825 {
00826     GtkFileChooserDialog *dlg;
00827     GtkFileFilter *filter1, *filter2;
00828     char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831     fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834     // Opening the saving dialog
00835     dlg = (GtkFileChooserDialog *)
00836         gtk_file_chooser_dialog_new (_("Save file"),
00837                                     window->window,
00838                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00840                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00841     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00842     buffer = g_build_filename (input->directory, input->name, NULL);
00843     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00844     g_free (buffer);
00845
00846     // Adding XML filter
00847     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00848     gtk_file_filter_set_name (filter1, "XML");
00849     gtk_file_filter_add_pattern (filter1, "*.xml");
00850     gtk_file_filter_add_pattern (filter1, "*.XML");
00851     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00852

```

```

00853 // Adding JSON filter
00854 filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00855 gtk_file_filter_set_name (filter2, "JSON");
00856 gtk_file_filter_add_pattern (filter2, "*.json");
00857 gtk_file_filter_add_pattern (filter2, "*.JSON");
00858 gtk_file_filter_add_pattern (filter2, "*.js");
00859 gtk_file_filter_add_pattern (filter2, "*.JS");
00860 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00861
00862 if (input->type == INPUT_TYPE_XML)
00863     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00864 else
00865     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00866
00867 // If OK response then saving
00868 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00869 {
00870     // Setting input file type
00871     filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00872     buffer = (char *) gtk_file_filter_get_name (filter1);
00873     if (!strcmp (buffer, "XML"))
00874         input->type = INPUT_TYPE_XML;
00875     else
00876         input->type = INPUT_TYPE_JSON;
00877
00878     // Adding properties to the root XML node
00879     input->simulator = gtk_file_chooser_get_filename
00880         (GTK_FILE_CHOOSER (window->button_simulator));
00881     if (gtk_toggle_button_get_active
00882         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00883         input->evaluator = gtk_file_chooser_get_filename
00884             (GTK_FILE_CHOOSER (window->button_evaluator));
00885     else
00886         input->evaluator = NULL;
00887     if (input->type == INPUT_TYPE_XML)
00888     {
00889         input->result
00890             = (char *) xmlStrdup ((const xmlChar *)
00891                 gtk_entry_get_text (window->entry_result));
00892         input->variables
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                 gtk_entry_get_text (window->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);
00935         input->nbest = gtk_spin_button_get_value_as_int (window->
00936 spin_bests);
00937         window_save_climbing ();
00938         break;
00939     default:

```

```

00933         input->algorithm = ALGORITHM_GENETIC;
00934         input->nsimulations
00935             = gtk_spin_button_get_value_as_int (window->spin_population);
00936         input->niterations
00937             = gtk_spin_button_get_value_as_int (window->spin_generations);
00938         input->mutation_ratio
00939             = gtk_spin_button_get_value (window->spin_mutation);
00940         input->reproduction_ratio
00941             = gtk_spin_button_get_value (window->spin_reproduction);
00942         input->adaptation_ratio
00943             = gtk_spin_button_get_value (window->spin_adaptation);
00944         break;
00945     }
00946     input->norm = window_get_norm ();
00947     input->p = gtk_spin_button_get_value (window->spin_p);
00948     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00949
00950     // Saving the XML file
00951     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00952     input_save (buffer);
00953
00954     // Closing and freeing memory
00955     g_free (buffer);
00956     gtk_widget_destroy (GTK_WIDGET (dlg));
00957 #if DEBUG_INTERFACE
00958     fprintf (stderr, "window_save: end\n");
00959 #endif
00960     return 1;
00961 }
00962
00963 // Closing and freeing memory
00964 gtk_widget_destroy (GTK_WIDGET (dlg));
00965 #if DEBUG_INTERFACE
00966     fprintf (stderr, "window_save: end\n");
00967 #endif
00968     return 0;
00969 }
00970
00971 static void
00972 window_run ()
00973 {
00974     unsigned int i;
00975     char *msg, *msg2, buffer[64], buffer2[64];
00976 #if DEBUG_INTERFACE
00977     fprintf (stderr, "window_run: start\n");
00978 #endif
00979     if (!window_save ())
00980     {
00981         #if DEBUG_INTERFACE
00982             fprintf (stderr, "window_run: end\n");
00983         #endif
00984         return;
00985     }
00986     running_new ();
00987     while (gtk_events_pending ())
00988         gtk_main_iteration ();
00989     optimize_open ();
00990 #if DEBUG_INTERFACE
00991     fprintf (stderr, "window_run: closing running dialog\n");
00992 #endif
00993     gtk_spinner_stop (running->spinner);
00994     gtk_widget_destroy (GTK_WIDGET (running->dialog));
00995 #if DEBUG_INTERFACE
00996     fprintf (stderr, "window_run: displaying results\n");
00997 #endif
00998     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00999     msg2 = g_strdup (buffer);
01000     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01001     {
01002         snprintf (buffer, 64, "%s = %s\n",
input->variable[i].name, format[input->
variable[i].precision]);
01003         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01004         msg = g_strconcat (msg2, buffer2, NULL);
01005         g_free (msg2);
01006     }
01007     snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
optimize->calculation_time);
01008     msg = g_strconcat (msg2, buffer, NULL);
01009     g_free (msg2);
01010     show_message (_("Best result"), msg, INFO_TYPE);
01011     g_free (msg);
01012 #if DEBUG_INTERFACE
01013     fprintf (stderr, "window_run: freeing memory\n");
01014 #endif
01015     optimize_free ();

```

```

01021 #if DEBUG_INTERFACE
01022     fprintf (stderr, "window_run: end\n");
01023 #endif
01024 }
01025
01029 static void
01030 window_help ()
01031 {
01032     char *buffer, *buffer2;
01033     #if DEBUG_INTERFACE
01034     fprintf (stderr, "window_help: start\n");
01035     #endif
01036     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01037                               _("user-manual.pdf"), NULL);
01038     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01039     g_free (buffer2);
01040     #if GTK_MINOR_VERSION >= 22
01041     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01042     #else
01043     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01044     #endif
01045     #if DEBUG_INTERFACE
01046     fprintf (stderr, "window_help: uri=%s\n", buffer);
01047     #endif
01048     g_free (buffer);
01049     #if DEBUG_INTERFACE
01050     fprintf (stderr, "window_help: end\n");
01051     #endif
01052 }
01053
01057 static void
01058 window_about ()
01059 {
01060     static const gchar *authors[] = {
01061         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01062         "Borja Latorre Garcés <borja.latorre@csic.es>",
01063         NULL
01064     };
01065     #if DEBUG_INTERFACE
01066     fprintf (stderr, "window_about: start\n");
01067     #endif
01068     gtk_show_about_dialog
01069     (window->window,
01070      "program_name", "MPCOTool",
01071      "comments",
01072      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01073       "A software to perform calibrations or optimizations of empirical "
01074       "parameters"),
01075      "authors", authors,
01076      "translator-credits",
01077      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01078      "(english, french and spanish)\n"
01079      "Uğur Çayoğlu (german)",
01080      "version", "4.0.1",
01081      "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
01082      "logo", window->logo,
01083      "website", "https://github.com/jburguete/mpcotool",
01084      "license-type", GTK_LICENSE_BSD, NULL);
01085     #if DEBUG_INTERFACE
01086     fprintf (stderr, "window_about: end\n");
01087     #endif
01088 }
01089
01093 static void
01094 window_update_climbing ()
01095 {
01096     #if DEBUG_INTERFACE
01097     fprintf (stderr, "window_update_climbing: start\n");
01098     #endif
01099     gtk_widget_show (GTK_WIDGET (window->check_climbing));
01100     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->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 }
01120
01124 static void
01125 window_update ()
01126 {
01127     unsigned int i;
01128 #if DEBUG_INTERFACE
01129     fprintf (stderr, "window_update: start\n");
01130 #endif
01131     gtk_widget_set_sensitive
01132         (GTK_WIDGET (window->button_evaluator),
01133          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01134                                         (window->check_evaluator)));
01135     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01136     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01137     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01138     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01139     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01140     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01141     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01142     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01143     gtk_widget_hide (GTK_WIDGET (window->label_population));
01144     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01145     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01146     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01147     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01148     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01149     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01150     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01151     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01152     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01153     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01154     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01155     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01156     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01157     gtk_widget_hide (GTK_WIDGET (window->check_climbing));
01158     gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
01159     gtk_widget_hide (GTK_WIDGET (window->label_step));
01160     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01161     gtk_widget_hide (GTK_WIDGET (window->label_p));
01162     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01163     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01164     switch (window_get_algorithm ())
01165     {
01166     case ALGORITHM_MONTE_CARLO:
01167         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01168         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01169         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01170         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01171         if (i > 1)
01172         {
01173             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01174             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01175             gtk_widget_show (GTK_WIDGET (window->label_bests));
01176             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01177         }
01178         window_update_climbing ();
01179         break;
01180     case ALGORITHM_SWEEP:
01181     case ALGORITHM_ORTHOGONAL:
01182         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01183         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01184         if (i > 1)
01185         {
01186             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01187             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01188             gtk_widget_show (GTK_WIDGET (window->label_bests));
01189             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01190         }
01191         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01192         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01193         gtk_widget_show (GTK_WIDGET (window->check_climbing));
01194         window_update_climbing ();
01195         break;
01196     default:
01197         gtk_widget_show (GTK_WIDGET (window->label_population));
01198         gtk_widget_show (GTK_WIDGET (window->spin_population));
01199         gtk_widget_show (GTK_WIDGET (window->label_generations));
01200         gtk_widget_show (GTK_WIDGET (window->spin_generations));
01201         gtk_widget_show (GTK_WIDGET (window->label_mutation));
01202         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01203         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01204         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01205         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01206         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));

```



```

01207     gtk_widget_show (GTK_WIDGET (window->label_bits));
01208     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01209 }
01210 gtk_widget_set_sensitive
01211 (GTK_WIDGET (window->button_remove_experiment),
input->nexperiments > 1);
01212 gtk_widget_set_sensitive
01213 (GTK_WIDGET (window->button_remove_variable), input->
nvariables > 1);
01214 for (i = 0; i < input->experiment->ninputs; ++i)
01215 {
01216     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01217     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01218     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01219     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01220     g_signal_handler_block
01221     (window->check_template[i], window->id_template[i]);
01222     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
01223     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_template[i]), 1);
01224     g_signal_handler_unblock (window->button_template[i],
window->id_input[i]);
01225     g_signal_handler_unblock (window->check_template[i],
window->id_template[i]);
01226     g_signal_handler_unblock (window->button_template[i],
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]),
gtk_toggle_button_get_active
GTK_TOGGLE_BUTTON (window->check_template
[i - 1]));
01232 }
01233 if (i < MAX_NINPUTS)
01234 {
01235     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01236     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01237     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01238     gtk_widget_set_sensitive
01239     (GTK_WIDGET (window->button_template[i]),
gtk_toggle_button_get_active
GTK_TOGGLE_BUTTON (window->check_template[i]));
01240     g_signal_handler_block
01241     (window->check_template[i], window->id_template[i]);
01242     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
01243     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_template[i]), 0);
01244     g_signal_handler_unblock (window->button_template[i],
window->id_input[i]);
01245     g_signal_handler_unblock (window->check_template[i],
window->id_template[i]);
01246 }
01247 while (++i < MAX_NINPUTS)
01248 {
01249     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01250     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01251 }
01252 gtk_widget_set_sensitive
01253 (GTK_WIDGET (window->spin_minabs),
gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01254 gtk_widget_set_sensitive
01255 (GTK_WIDGET (window->spin_maxabs),
gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01256 if (window_get_norm () == ERROR_NORM_P)
01257 {
01258     gtk_widget_show (GTK_WIDGET (window->label_p));
01259     gtk_widget_show (GTK_WIDGET (window->spin_p));
01260 }
01261 #if DEBUG_INTERFACE
01262 fprintf (stderr, "window_update: end\n");
01263 #endif
01264 }
01265 static void
01266 window_set_algorithm ()
01267 {
01268     int i;
01269     #if DEBUG_INTERFACE
01270     fprintf (stderr, "window_set_algorithm: start\n");
01271     #endif
01272     i = window_get_algorithm ();
01273     switch (i)
01274     {
01275     case ALGORITHM_SWEEP:
01276     case ALGORITHM_ORTHOGONAL:

```

```

01293     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01294     if (i < 0)
01295         i = 0;
01296     gtk_spin_button_set_value (window->spin_sweeps,
01297                               (gdouble) input->variable[i].
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
01315 static void
01316 window_set_experiment ()
01317 {
01318     unsigned int i, j;
01319     char *buffer1, *buffer2;
01320 #if DEBUG_INTERFACE
01321     fprintf (stderr, "window_set_experiment: start\n");
01322 #endif
01323     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01324     gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
01325     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01326     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01327     g_free (buffer1);
01328     g_signal_handler_block
01329         (window->button_experiment, window->id_experiment_name);
01330     gtk_file_chooser_set_filename
01331         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01332     g_signal_handler_unblock
01333         (window->button_experiment, window->id_experiment_name);
01334     g_free (buffer2);
01335     for (j = 0; j < input->experiment->ninputs; ++j)
01336     {
01337         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01338         buffer2 =
01339             g_build_filename (input->directory, input->experiment[i].
stencil[j],
01340                             NULL);
01341         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01342                                     (window->button_template[j]), buffer2);
01343         g_free (buffer2);
01344         g_signal_handler_unblock
01345             (window->button_template[j], window->id_input[j]);
01346     }
01347 #if DEBUG_INTERFACE
01348     fprintf (stderr, "window_set_experiment: end\n");
01349 #endif
01350 }
01351
01355 static void
01356 window_remove_experiment ()
01357 {
01358     unsigned int i, j;
01359 #if DEBUG_INTERFACE
01360     fprintf (stderr, "window_remove_experiment: start\n");
01361 #endif
01362     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01363     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01364     gtk_combo_box_text_remove (window->combo_experiment, i);
01365     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01366     experiment_free (input->experiment + i, input->
type);
01367     --input->nexperiments;
01368     for (j = i; j < input->nexperiments; ++j)
01369         memcpy (input->experiment + j, input->experiment + j + 1,
01370               sizeof (Experiment));
01371     j = input->nexperiments - 1;
01372     if (i > j)
01373         i = j;
01374     for (j = 0; j < input->experiment->ninputs; ++j)
01375         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01376     g_signal_handler_block
01377         (window->button_experiment, window->id_experiment_name);

```

```

01378     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01379     g_signal_handler_unblock
01380     (window->button_experiment, window->id_experiment_name);
01381     for (j = 0; j < input->experiment->ninputs; ++j)
01382         g_signal_handler_unblock (window->button_template[j], window->
01383         id_input[j]);
01384     window_update ();
01385     #if DEBUG_INTERFACE
01386     fprintf (stderr, "window_remove_experiment: end\n");
01387     #endif
01388 }
01389
01390 static void
01391 window_add_experiment ()
01392 {
01393     unsigned int i, j;
01394     #if DEBUG_INTERFACE
01395     fprintf (stderr, "window_add_experiment: start\n");
01396     #endif
01397     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01398     g_signal_handler_block (window->combo_experiment, window->
01399     id_experiment);
01400     gtk_combo_box_text_insert_text
01401     (window->combo_experiment, i, input->experiment[i].
01402     name);
01403     g_signal_handler_unblock (window->combo_experiment, window->
01404     id_experiment);
01405     input->experiment = (Experiment *) g_realloc
01406     (input->experiment, (input->nexperiments + 1) * sizeof (
01407     Experiment));
01408     for (j = input->nexperiments - 1; j > i; --j)
01409         mempcpy (input->experiment + j + 1, input->experiment + j,
01410         sizeof (Experiment));
01411     input->experiment[j + 1].weight = input->experiment[j].
01412     weight;
01413     input->experiment[j + 1].ninputs = input->
01414     experiment[j].ninputs;
01415     if (input->type == INPUT_TYPE_XML)
01416     {
01417         input->experiment[j + 1].name
01418         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
01419         name);
01420         for (j = 0; j < input->experiment->ninputs; ++j)
01421             input->experiment[i + 1].stencil[j]
01422             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
01423             stencil[j]);
01424     }
01425     else
01426     {
01427         input->experiment[j + 1].name = g_strdup (input->
01428         experiment[j].name);
01429         for (j = 0; j < input->experiment->ninputs; ++j)
01430             input->experiment[i + 1].stencil[j]
01431             = g_strdup (input->experiment[i].stencil[j]);
01432     }
01433     ++input->nexperiments;
01434     for (j = 0; j < input->experiment->ninputs; ++j)
01435         g_signal_handler_block (window->button_template[j], window->
01436         id_input[j]);
01437     g_signal_handler_block
01438     (window->button_experiment, window->id_experiment_name);
01439     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01440     g_signal_handler_unblock
01441     (window->button_experiment, window->id_experiment_name);
01442     for (j = 0; j < input->experiment->ninputs; ++j)
01443         g_signal_handler_unblock (window->button_template[j], window->
01444         id_input[j]);
01445     window_update ();
01446     #if DEBUG_INTERFACE
01447     fprintf (stderr, "window_add_experiment: end\n");
01448     #endif
01449 }
01450
01451 static void
01452 window_name_experiment ()
01453 {
01454     unsigned int i;
01455     char *buffer;
01456     GFile *file1, *file2;
01457     #if DEBUG_INTERFACE
01458     fprintf (stderr, "window_name_experiment: start\n");
01459     #endif
01460     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01461     file1
01462     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01463     file2 = g_file_new_for_path (input->directory);
01464     buffer = g_file_get_relative_path (file2, file1);

```

```

01459 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01460 gtk_combo_box_text_remove (window->combo_experiment, i);
01461 gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01462 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01463 g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01464 g_free (buffer);
01465 g_object_unref (file2);
01466 g_object_unref (file1);
01467 #if DEBUG_INTERFACE
01468 fprintf (stderr, "window_name_experiment: end\n");
01469 #endif
01470 }
01471
01475 static void
01476 window_weight_experiment ()
01477 {
01478     unsigned int i;
01479     #if DEBUG_INTERFACE
01480     fprintf (stderr, "window_weight_experiment: start\n");
01481     #endif
01482     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01483     input->experiment[i].weight = gtk_spin_button_get_value (window->
spin_weight);
01484     #if DEBUG_INTERFACE
01485     fprintf (stderr, "window_weight_experiment: end\n");
01486     #endif
01487 }
01488
01492 static void
01493 window_inputs_experiment ()
01494 {
01495     unsigned int j;
01496     #if DEBUG_INTERFACE
01497     fprintf (stderr, "window_inputs_experiment: start\n");
01498     #endif
01499     j = input->experiment->ninputs - 1;
01500     if (j
        && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
            (window->check_template[j])))
01501         --input->experiment->ninputs;
01502     if (input->experiment->ninputs < MAX_NINPUTS
        && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
            (window->check_template[j])))
01503         ++input->experiment->ninputs;
01504     window_update ();
01505     #if DEBUG_INTERFACE
01506     fprintf (stderr, "window_inputs_experiment: end\n");
01507     #endif
01508 }
01509
01517 static void
01518 window_template_experiment (void *data)
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 }
01542
01548 static void
01549 window_set_variable ()
01550 {
01551     unsigned int i;
01552     #if DEBUG_INTERFACE
01553     fprintf (stderr, "window_set_variable: start\n");
01554     #endif
01555     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));

```

```

01556 g_signal_handler_block (window->entry_variable, window->
01557 id_variable_label);
01557 gtk_entry_set_text (window->entry_variable, input->variable[i].
01558 name);
01558 g_signal_handler_unblock (window->entry_variable, window->
01559 id_variable_label);
01559 gtk_spin_button_set_value (window->spin_min, input->variable[i].
01560 rangemin);
01560 gtk_spin_button_set_value (window->spin_max, input->variable[i].
01561 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 {
01583     gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01584     gtk_toggle_button_set_active
01585     (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01586 }
01587 gtk_spin_button_set_value (window->spin_precision,
01588                             input->variable[i].precision);
01589 gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01590 nsteps);
01590 if (input->nsteps)
01591     gtk_spin_button_set_value (window->spin_step, input->variable[i].
01592 step);
01592 #if DEBUG_INTERFACE
01593 fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01594         input->variable[i].precision);
01595 #endif
01596 switch (window_get_algorithm ())
01597 {
01598     case ALGORITHM_SWEEP:
01599     case ALGORITHM_ORTHOGONAL:
01600         gtk_spin_button_set_value (window->spin_sweeps,
01601                                     (gdouble) input->variable[i].
01602 nsweeps);
01603 #if DEBUG_INTERFACE
01604 fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01605         input->variable[i].nsweeps);
01606 #endif
01607         break;
01608     case ALGORITHM_GENETIC:
01609         gtk_spin_button_set_value (window->spin_bits,
01610                                     (gdouble) input->variable[i].nbits);
01611 #if DEBUG_INTERFACE
01612 fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01613         input->variable[i].nbits);
01614 #endif
01615         break;
01616 }
01616 window_update ();
01617 #if DEBUG_INTERFACE
01618 fprintf (stderr, "window_set_variable: end\n");
01619 #endif
01620 }
01621
01622 static void
01623 window_remove_variable ()
01624 {
01625     unsigned int i, j;
01626     #if DEBUG_INTERFACE
01627     fprintf (stderr, "window_remove_variable: start\n");
01628     #endif
01629     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01630     g_signal_handler_block (window->combo_variable, window->
01631 id_variable);
01632     gtk_combo_box_text_remove (window->combo_variable, i);
01633     g_signal_handler_unblock (window->combo_variable, window->
01634 id_variable);

```

```

01636     xmlFree (input->variable[i].name);
01637     --input->nvariables;
01638     for (j = i; j < input->nvariables; ++j)
01639         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01640     j = input->nvariables - 1;
01641     if (i > j)
01642         i = j;
01643     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01644     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01645     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01646     window_update ();
01647     #if DEBUG_INTERFACE
01648     fprintf (stderr, "window_remove_variable: end\n");
01649     #endif
01650 }
01651
01652 static void
01653 window_add_variable ()
01654 {
01655     unsigned int i, j;
01656     #if DEBUG_INTERFACE
01657     fprintf (stderr, "window_add_variable: start\n");
01658     #endif
01659     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01660     g_signal_handler_block (window->combo_variable, window->
id_variable_label);
01661     gtk_combo_box_text_insert_text (window->combo_variable, i,
input->variable[i].name);
01662     g_signal_handler_unblock (window->combo_variable, window->
id_variable_label);
01663     input->variable = (Variable *) g_realloc
(input->variable, (input->nvariables + 1) * sizeof (
Variable));
01664     for (j = input->nvariables - 1; j > i; --j)
01665         memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01666     memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01667     if (input->type == INPUT_TYPE_XML)
01668         input->variable[j + 1].name
= (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01669     else
01670         input->variable[j + 1].name = g_strdup (input->
variable[j].name);
01671     ++input->nvariables;
01672     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01673     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01674     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01675     window_update ();
01676     #if DEBUG_INTERFACE
01677     fprintf (stderr, "window_add_variable: end\n");
01678     #endif
01679 }
01680
01681 static void
01682 window_label_variable ()
01683 {
01684     unsigned int i;
01685     const char *buffer;
01686     #if DEBUG_INTERFACE
01687     fprintf (stderr, "window_label_variable: start\n");
01688     #endif
01689     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01690     buffer = gtk_entry_get_text (window->entry_variable);
01691     g_signal_handler_block (window->combo_variable, window->
id_variable_label);
01692     gtk_combo_box_text_remove (window->combo_variable, i);
01693     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01694     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01695     g_signal_handler_unblock (window->combo_variable, window->
id_variable_label);
01696     #if DEBUG_INTERFACE
01697     fprintf (stderr, "window_label_variable: end\n");
01698     #endif
01699 }
01700
01701 static void
01702 window_precision_variable ()
01703 {
01704     unsigned int i;
01705     #if DEBUG_INTERFACE
01706     fprintf (stderr, "window_precision_variable: start\n");

```

```

01719 #endif
01720 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01721 input->variable[i].precision
01722 = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01723 gtk_spin_button_set_digits (window->spin_min, input->variable[i].
precision);
01724 gtk_spin_button_set_digits (window->spin_max, input->variable[i].
precision);
01725 gtk_spin_button_set_digits (window->spin_minabs,
01726 input->variable[i].precision);
01727 gtk_spin_button_set_digits (window->spin_maxabs,
01728 input->variable[i].precision);
01729 #if DEBUG_INTERFACE
01730 fprintf (stderr, "window_precision_variable: end\n");
01731 #endif
01732 }
01733
01737 static void
01738 window_rangemin_variable ()
01739 {
01740 unsigned int i;
01741 #if DEBUG_INTERFACE
01742 fprintf (stderr, "window_rangemin_variable: start\n");
01743 #endif
01744 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01745 input->variable[i].rangemin = gtk_spin_button_get_value (window->
spin_min);
01746 #if DEBUG_INTERFACE
01747 fprintf (stderr, "window_rangemin_variable: end\n");
01748 #endif
01749 }
01750
01754 static void
01755 window_rangemax_variable ()
01756 {
01757 unsigned int i;
01758 #if DEBUG_INTERFACE
01759 fprintf (stderr, "window_rangemax_variable: start\n");
01760 #endif
01761 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01762 input->variable[i].rangemax = gtk_spin_button_get_value (window->
spin_max);
01763 #if DEBUG_INTERFACE
01764 fprintf (stderr, "window_rangemax_variable: end\n");
01765 #endif
01766 }
01767
01771 static void
01772 window_rangeminabs_variable ()
01773 {
01774 unsigned int i;
01775 #if DEBUG_INTERFACE
01776 fprintf (stderr, "window_rangeminabs_variable: start\n");
01777 #endif
01778 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01779 input->variable[i].rangeminabs
01780 = gtk_spin_button_get_value (window->spin_minabs);
01781 #if DEBUG_INTERFACE
01782 fprintf (stderr, "window_rangeminabs_variable: end\n");
01783 #endif
01784 }
01785
01789 static void
01790 window_rangemaxabs_variable ()
01791 {
01792 unsigned int i;
01793 #if DEBUG_INTERFACE
01794 fprintf (stderr, "window_rangemaxabs_variable: start\n");
01795 #endif
01796 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01797 input->variable[i].rangemaxabs
01798 = gtk_spin_button_get_value (window->spin_maxabs);
01799 #if DEBUG_INTERFACE
01800 fprintf (stderr, "window_rangemaxabs_variable: end\n");
01801 #endif
01802 }
01803
01807 static void
01808 window_step_variable ()
01809 {
01810 unsigned int i;
01811 #if DEBUG_INTERFACE
01812 fprintf (stderr, "window_step_variable: start\n");
01813 #endif
01814 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01815 input->variable[i].step = gtk_spin_button_get_value (window->
spin_step);

```

```

01816 #if DEBUG_INTERFACE
01817     fprintf (stderr, "window_step_variable: end\n");
01818 #endif
01819 }
01820
01824 static void
01825 window_update_variable ()
01826 {
01827     int i;
01828 #if DEBUG_INTERFACE
01829     fprintf (stderr, "window_update_variable: start\n");
01830 #endif
01831     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01832     if (i < 0)
01833         i = 0;
01834     switch (window_get_algorithm ())
01835     {
01836     case ALGORITHM_SWEEP:
01837     case ALGORITHM_ORTHOGONAL:
01838         input->variable[i].nsweeps
01839         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01840 #if DEBUG_INTERFACE
01841         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01842                 input->variable[i].nsweeps);
01843 #endif
01844         break;
01845     case ALGORITHM_GENETIC:
01846         input->variable[i].nbits
01847         = gtk_spin_button_get_value_as_int (window->spin_bits);
01848 #if DEBUG_INTERFACE
01849         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01850                 input->variable[i].nbits);
01851 #endif
01852     }
01853 #if DEBUG_INTERFACE
01854     fprintf (stderr, "window_update_variable: end\n");
01855 #endif
01856 }
01857
01863 static int
01864 window_read (char *filename)
01865 {
01866     unsigned int i;
01867     char *buffer;
01868 #if DEBUG_INTERFACE
01869     fprintf (stderr, "window_read: start\n");
01870 #endif
01871
01872     // Reading new input file
01873     input_free ();
01874     input->result = input->variables = NULL;
01875     if (!input_open (filename))
01876     {
01877 #if DEBUG_INTERFACE
01878         fprintf (stderr, "window_read: end\n");
01879 #endif
01880         return 0;
01881     }
01882
01883     // Setting GTK+ widgets data
01884     gtk_entry_set_text (window->entry_result, input->result);
01885     gtk_entry_set_text (window->entry_variables, input->
variables);
01886     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01887     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01888     g_free (buffer);
01889     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01890
01891     if (input->evaluator)
01892     {
01893         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01894         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01895         g_free (buffer);
01896     }
01897     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01901     switch (input->algorithm)
01902     {
01903     case ALGORITHM_MONTE_CARLO:
01904         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01905
01906         // fallthrough

```



```

01907     case ALGORITHM_SWEEP:
01908     case ALGORITHM_ORTHOGONAL:
01909         gtk_spin_button_set_value (window->spin_iterations,
01910             (gdouble) input->niterations);
01911         gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01912         gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01913         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01914             (window->check_climbing),
input->nsteps);
01915         if (input->nsteps)
01916         {
01917             gtk_toggle_button_set_active
01918                 (GTK_TOGGLE_BUTTON (window->button_climbing[input->
climbing]),
TRUE);
01919             gtk_spin_button_set_value (window->spin_steps,
01920                 (gdouble) input->nsteps);
01921             gtk_spin_button_set_value (window->spin_relaxation,
01922                 (gdouble) input->relaxation);
01923             switch (input->climbing)
01924             {
01925             case CLIMBING_METHOD_RANDOM:
01926                 gtk_spin_button_set_value (window->spin_estimates,
01927                     (gdouble) input->nestimates);
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,
01937                 input->reproduction_ratio);
01938             gtk_spin_button_set_value (window->spin_adaptation,
01939                 input->adaptation_ratio);
01940         }
01941         gtk_toggle_button_set_active
01942             (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01943         gtk_spin_button_set_value (window->spin_p, input->p);
01944         gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01945         g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01946         g_signal_handler_block (window->button_experiment,
01947             window->id_experiment_name);
01948         gtk_combo_box_text_remove_all (window->combo_experiment);
01949         for (i = 0; i < input->nexperiments; ++i)
01950             gtk_combo_box_text_append_text (window->combo_experiment,
01951                 input->experiment[i].name);
01952         g_signal_handler_unblock
01953             (window->button_experiment, window->id_experiment_name);
01954         g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01955         gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01956         g_signal_handler_block (window->combo_variable, window->
id_variable);
01957         g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01958         gtk_combo_box_text_remove_all (window->combo_variable);
01959         for (i = 0; i < input->nvariables; ++i)
01960             gtk_combo_box_text_append_text (window->combo_variable,
01961                 input->variable[i].name);
01962         g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01963         g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01964         gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01965         window_set_variable ();
01966         window_update ();
01967     #if DEBUG_INTERFACE
01968         fprintf (stderr, "window_read: end\n");
01969     #endif
01970     return 1;
01971 }
01972
01973 static void
01974 window_open ()
01975 {
01976     GtkFileChooserDialog *dlg;
01977     GtkFileFilter *filter;
01978     char *buffer, *directory, *name;

```

```

01985
01986 #if DEBUG_INTERFACE
01987     fprintf (stderr, "window_open: start\n");
01988 #endif
01989
01990 // Saving a backup of the current input file
01991 directory = g_strdup (input->directory);
01992 name = g_strdup (input->name);
01993
01994 // Opening dialog
01995 dlg = (GtkFileChooserDialog *)
01996     gtk_file_chooser_dialog_new (_, "Open input file",
01997         window->window,
01998         GTK_FILE_CHOOSER_ACTION_OPEN,
01999         _("_Cancel"), GTK_RESPONSE_CANCEL,
02000         _("_OK"), GTK_RESPONSE_OK, NULL);
02001
02002 // Adding XML filter
02003 filter = (GtkFileFilter *) gtk_file_filter_new ();
02004 gtk_file_filter_set_name (filter, "XML");
02005 gtk_file_filter_add_pattern (filter, "*.xml");
02006 gtk_file_filter_add_pattern (filter, "*.XML");
02007 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02008
02009 // Adding JSON filter
02010 filter = (GtkFileFilter *) gtk_file_filter_new ();
02011 gtk_file_filter_set_name (filter, "JSON");
02012 gtk_file_filter_add_pattern (filter, "*.json");
02013 gtk_file_filter_add_pattern (filter, "*.JSON");
02014 gtk_file_filter_add_pattern (filter, "*.js");
02015 gtk_file_filter_add_pattern (filter, "*.JS");
02016 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02017
02018 // If OK saving
02019 while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
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             // Closing on backup file reading error
02036             #if DEBUG_INTERFACE
02037                 fprintf (stderr, "window_read: error reading backup file\n");
02038             #endif
02039             g_free (buffer);
02040             break;
02041         }
02042         g_free (buffer);
02043     }
02044     else
02045     {
02046         g_free (buffer);
02047         break;
02048     }
02049 }
02050
02051 // Freeing and closing
02052 g_free (name);
02053 g_free (directory);
02054 gtk_widget_destroy (GTK_WIDGET (dlg));
02055 #if DEBUG_INTERFACE
02056     fprintf (stderr, "window_open: end\n");
02057 #endif
02058 }
02059
02060 void
02061 window_new (GtkApplication * application)
02062 {
02063     unsigned int i;
02064     char *buffer, *buffer2, buffer3[64];
02065     char *label_algorithm[NALGORITHMMS] = {
02066         "Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02067     };
02068     char *tip_algorithm[NALGORITHMMS] = {
02069         "Monte-Carlo brute force algorithm",

```

```

02075     _("Sweep brute force algorithm"),
02076     _("Genetic algorithm"),
02077     _("Orthogonal sampling brute force algorithm"),
02078 };
02079 char *label_climbing[NCLIMBINGS] = {
02080     _("_Coordinates climbing"), _("_Random climbing")
02081 };
02082 char *tip_climbing[NCLIMBINGS] = {
02083     _("Coordinates climbing estimate method"),
02084     _("Random climbing estimate method")
02085 };
02086 char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02087 char *tip_norm[NNORMS] = {
02088     _("Euclidean error norm (L2)"),
02089     _("Maximum error norm (L)"),
02090     _("P error norm (Lp)"),
02091     _("Taxicab error norm (L1)")
02092 };
02093
02094 #if DEBUG_INTERFACE
02095     fprintf (stderr, "window_new: start\n");
02096 #endif
02097
02098 // Creating the window
02099 window->window = main_window
02100     = (GtkWindow *) gtk_application_window_new (application);
02101
02102 // Finish when closing the window
02103 g_signal_connect_swapped (window->window, "delete-event",
02104     G_CALLBACK (g_application_quit),
02105     G_APPLICATION (application));
02106
02107 // Setting the window title
02108 gtk_window_set_title (window->window, "MPCOTool");
02109
02110 // Creating the open button
02111 window->button_open = (GtkToolButton *) gtk_tool_button_new
02112     (gtk_image_new_from_icon_name ("document-open",
02113         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02114 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02115
02116 // Creating the save button
02117 window->button_save = (GtkToolButton *) gtk_tool_button_new
02118     (gtk_image_new_from_icon_name ("document-save",
02119         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02120 g_signal_connect (window->button_save, "clicked", (GCallback)
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", options_new, NULL);
02134
02135 // Creating the help button
02136 window->button_help = (GtkToolButton *) gtk_tool_button_new
02137     (gtk_image_new_from_icon_name ("help-browser",
02138         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02139 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02140
02141 // Creating the about button
02142 window->button_about = (GtkToolButton *) gtk_tool_button_new
02143     (gtk_image_new_from_icon_name ("help-about",
02144         GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02145 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02146
02147 // Creating the exit button
02148 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02149     (gtk_image_new_from_icon_name ("application-exit",
02150         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02151 g_signal_connect_swapped (window->button_exit, "clicked",
02152     G_CALLBACK (g_application_quit),
02153     G_APPLICATION (application));
02154
02155 // Creating the buttons bar
02156 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02157 gtk_toolbar_insert
02158     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02159 gtk_toolbar_insert
02160     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);

```

```

02161 gtk_toolbar_insert
02162     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02163 gtk_toolbar_insert
02164     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02165 gtk_toolbar_insert
02166     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02167 gtk_toolbar_insert
02168     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02169 gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02171 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02172
02173 // Creating the simulator program label and entry
02174 window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02175 window->button_simulator = (GtkFileChooserButton *)
02176     gtk_file_chooser_button_new (_("Simulator program"),
02177     GTK_FILE_CHOOSER_ACTION_OPEN);
02178 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02179     _("Simulator program executable file"));
02180 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02181
02182 // Creating the evaluator program label and entry
02183 window->check_evaluator = (GtkCheckButton *)
02184     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02185 g_signal_connect (window->check_evaluator, "toggled",
02186     window_update, NULL);
02187 window->button_evaluator = (GtkFileChooserButton *)
02188     gtk_file_chooser_button_new (_("Evaluator program"),
02189     GTK_FILE_CHOOSER_ACTION_OPEN);
02190 gtk_widget_set_tooltip_text
02191     (GTK_WIDGET (window->button_evaluator),
02192     _("Optional evaluator program executable file"));
02193
02194 // Creating the results files labels and entries
02195 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02196 window->entry_result = (GtkEntry *) gtk_entry_new ();
02197 gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->entry_result), _("Best results file"));
02199 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02200 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02201 gtk_widget_set_tooltip_text
02202     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02203
02204 // Creating the files grid and attaching widgets
02205 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02206     label_simulator),
02207     0, 0, 1, 1);
02208 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02209     button_simulator),
02210     1, 0, 1, 1);
02211 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02212     check_evaluator),
02213     0, 1, 1, 1);
02214 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02215     button_evaluator),
02216     1, 1, 1, 1);
02217 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02218     label_result),
02219     0, 2, 1, 1);
02220 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02221     entry_result),
02222     1, 2, 1, 1);
02223 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02224     label_variables),
02225     0, 3, 1, 1);
02226 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02227     entry_variables),
02228     1, 3, 1, 1);
02229
02230 // Creating the algorithm properties
02231 window->label_simulations = (GtkLabel *) gtk_label_new
02232     (_("Simulations number"));
02233 window->spin_simulations
02234     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02235 gtk_widget_set_tooltip_text
02236     (GTK_WIDGET (window->spin_simulations),
02237     _("Number of simulations to perform for each iteration"));
02238 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02239 window->label_iterations = (GtkLabel *)
02240     gtk_label_new (_("Iterations number"));
02241 window->spin_iterations
02242     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02243 gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02245 g_signal_connect
02246     (window->spin_iterations, "value-changed", window_update, NULL);

```

```

02239 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02240 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02241 window->spin_tolerance =
02242     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02243 gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->spin_tolerance),
02245      _("Tolerance to set the variable interval on the next iteration"));
02246 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02247 window->spin_bests
02248     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02249 gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->spin_bests),
02251      _("Number of best simulations used to set the variable interval "
02252        "on the next iteration"));
02253 window->label_population
02254     = (GtkLabel *) gtk_label_new (_("Population number"));
02255 window->spin_population
02256     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02257 gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_population),
02259      _("Number of population for the genetic algorithm"));
02260 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02261 window->label_generations
02262     = (GtkLabel *) gtk_label_new (_("Generations number"));
02263 window->spin_generations
02264     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02265 gtk_widget_set_tooltip_text
02266     (GTK_WIDGET (window->spin_generations),
02267      _("Number of generations for the genetic algorithm"));
02268 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02269 window->spin_mutation
02270     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02271 gtk_widget_set_tooltip_text
02272     (GTK_WIDGET (window->spin_mutation),
02273      _("Ratio of mutation for the genetic algorithm"));
02274 window->label_reproduction
02275     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02276 window->spin_reproduction
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02278 gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_reproduction),
02280      _("Ratio of reproduction for the genetic algorithm"));
02281 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02282 window->spin_adaptation
02283     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02284 gtk_widget_set_tooltip_text
02285     (GTK_WIDGET (window->spin_adaptation),
02286      _("Ratio of adaptation for the genetic algorithm"));
02287 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02288 window->spin_threshold = (GtkSpinButton *)
02289     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02290     precision[DEFAULT_PRECISION]);
02291 gtk_widget_set_tooltip_text
02292     (GTK_WIDGET (window->spin_threshold),
02293      _("Threshold in the objective function to finish the simulations"));
02294 window->scrolled_threshold =
02295     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02296 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02297     GTK_WIDGET (window->spin_threshold));
02298 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02299 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02300 //     GTK_ALIGN_FILL);
02301
02302 // Creating the hill climbing method properties
02303 window->check_climbing = (GtkCheckButton *)
02304     gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
02305 g_signal_connect (window->check_climbing, "clicked",
02306     window_update, NULL);
02307 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02308 window->button_climbing[0] = (GtkRadioButton *)
02309     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02310 gtk_grid_attach (window->grid_climbing,
02311     GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02312 g_signal_connect (window->button_climbing[0], "clicked",
02313     window_update, NULL);
02314 for (i = 0; ++i < NCLIMBINGS;)
02315 {
02316     window->button_climbing[i] = (GtkRadioButton *)
02317         gtk_radio_button_new_with_mnemonic
02318             (gtk_radio_button_get_group (window->button_climbing[0]),
02319              label_climbing[i]);
02320     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02321         tip_climbing[i]);
02322     gtk_grid_attach (window->grid_climbing,
02323         GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02324     g_signal_connect (window->button_climbing[i], "clicked",
02325         window_update,

```

```

02323             NULL);
02324         }
02325         window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02326         window->spin_steps = (GtkSpinButton *)
02327             gtk_spin_button_new_with_range (1., 1.e12, 1.);
02328         gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02329         window->label_estimates
02330             = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02331         window->spin_estimates = (GtkSpinButton *)
02332             gtk_spin_button_new_with_range (1., 1.e3, 1.);
02333         window->label_relaxation
02334             = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02335         window->spin_relaxation = (GtkSpinButton *)
02336             gtk_spin_button_new_with_range (0., 2., 0.001);
02337         gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_steps),
02338             0, NCLIMBINGS, 1, 1);
02339         gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_steps),
02340             1, NCLIMBINGS, 1, 1);
02341         gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_estimates),
02342             0, NCLIMBINGS + 1, 1, 1);
02343         gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_estimates),
02344             1, NCLIMBINGS + 1, 1, 1);
02345         gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_relaxation),
02346             0, NCLIMBINGS + 2, 1, 1);
02347         gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_relaxation),
02348             1, NCLIMBINGS + 2, 1, 1);
02349
02350         // Creating the array of algorithms
02351         window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02352         window->button_algorithm[0] = (GtkRadioButton *)
02353             gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02354         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02355             tip_algorithm[0]);
02356         gtk_grid_attach (window->grid_algorithm,
02357             GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02358         g_signal_connect (window->button_algorithm[0], "clicked",
02359             window_set_algorithm, NULL);
02360         for (i = 0; ++i < NALGORITHMS;)
02361         {
02362             window->button_algorithm[i] = (GtkRadioButton *)
02363                 gtk_radio_button_new_with_mnemonic
02364                 (gtk_radio_button_get_group (window->button_algorithm[0]),
02365                     label_algorithm[i]);
02366             gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02367                 tip_algorithm[i]);
02368             gtk_grid_attach (window->grid_algorithm,
02369                 GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02370             g_signal_connect (window->button_algorithm[i], "clicked",
02371                 window_set_algorithm, NULL);
02372         }
02373         gtk_grid_attach (window->grid_algorithm,
02374             GTK_WIDGET (window->label_simulations),
02375             0, NALGORITHMS, 1, 1);
02376         gtk_grid_attach (window->grid_algorithm,
02377             GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02378         gtk_grid_attach (window->grid_algorithm,
02379             GTK_WIDGET (window->label_iterations),
02380             0, NALGORITHMS + 1, 1, 1);
02381         gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_iterations),
02382             1, NALGORITHMS + 1, 1, 1);
02383         gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_tolerance),
02384             0, NALGORITHMS + 2, 1, 1);
02385         gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_tolerance),
02386             1, NALGORITHMS + 2, 1, 1);
02387         gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_bests),
02388             0, NALGORITHMS + 3, 1, 1);
02389         gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_bests),
02390             1, NALGORITHMS + 3, 1, 1);
02391         gtk_grid_attach (window->grid_algorithm,
02392             GTK_WIDGET (window->label_population),
02393             0, NALGORITHMS + 4, 1, 1);
02394         gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_population),
02395             1, NALGORITHMS + 4, 1, 1);
02396         gtk_grid_attach (window->grid_algorithm,
02397             GTK_WIDGET (window->label_generations),

```

```

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

```



```

02478     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02479     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02480     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02481     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02482     gtk_widget_set_tooltip_text
02483     (GTK_WIDGET (window->spin_minabs),
02484      _("Minimum allowed value of the variable"));
02485     window->scrolled_minabs
02486     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02487     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02488                       GTK_WIDGET (window->spin_minabs));
02489     g_signal_connect (window->spin_minabs, "value-changed",
02490                       window_rangeminabs_variable, NULL);
02491     window->check_maxabs = (GtkCheckButton *)
02492     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02493     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02494     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02495     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02496     gtk_widget_set_tooltip_text
02497     (GTK_WIDGET (window->spin_maxabs),
02498      _("Maximum allowed value of the variable"));
02499     window->scrolled_maxabs
02500     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02501     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02502                       GTK_WIDGET (window->spin_maxabs));
02503     g_signal_connect (window->spin_maxabs, "value-changed",
02504                       window_rangemaxabs_variable, NULL);
02505     window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02506     window->spin_precision = (GtkSpinButton *)
02507     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02508     gtk_widget_set_tooltip_text
02509     (GTK_WIDGET (window->spin_precision),
02510      _("Number of precision floating point digits\n"
02511        "0 is for integer numbers"));
02512     g_signal_connect (window->spin_precision, "value-changed",
02513                       window_precision_variable, NULL);
02514     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02515     window->spin_sweeps =
02516     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02517     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02518      _("Number of steps sweeping the variable"));
02519     g_signal_connect (window->spin_sweeps, "value-changed",
02520                       window_update_variable, NULL);
02521     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02522     window->spin_bits
02523     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02524     gtk_widget_set_tooltip_text
02525     (GTK_WIDGET (window->spin_bits),
02526      _("Number of bits to encode the variable"));
02527     g_signal_connect
02528     (window->spin_bits, "value-changed", window_update_variable, NULL);
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);
02558     gtk_grid_attach (window->grid_variable,
02559                     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02560     gtk_grid_attach (window->grid_variable,
02561                     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02562     gtk_grid_attach (window->grid_variable,
02563                     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02564     gtk_grid_attach (window->grid_variable,

```



```

02565         GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02566     gtk_grid_attach (window->grid_variable,
02567         GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02568     gtk_grid_attach (window->grid_variable,
02569         GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02570     gtk_grid_attach (window->grid_variable,
02571         GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02572     gtk_grid_attach (window->grid_variable,
02573         GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02574     gtk_grid_attach (window->grid_variable,
02575         GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02576     gtk_grid_attach (window->grid_variable,
02577         GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02578     gtk_grid_attach (window->grid_variable,
02579         GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02580     gtk_grid_attach (window->grid_variable,
02581         GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02582     gtk_grid_attach (window->grid_variable,
02583         GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02584     window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02585     gtk_container_add (GTK_CONTAINER (window->frame_variable),
02586         GTK_WIDGET (window->grid_variable));
02587
02588     // Creating the experiment widgets
02589     window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02590     gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02591         _("Experiment selector"));
02592     window->id_experiment = g_signal_connect
02593         (window->combo_experiment, "changed", window_set_experiment, NULL)
02594 ;
02595     window->button_add_experiment = (GtkButton *)
02596         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02597     g_signal_connect
02598         (window->button_add_experiment, "clicked",
02599         window_add_experiment, NULL);
02600     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02601         _("Add experiment"));
02602     window->button_remove_experiment = (GtkButton *)
02603         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02604     g_signal_connect (window->button_remove_experiment, "clicked",
02605         window_remove_experiment, NULL);
02606     gtk_widget_set_tooltip_text (GTK_WIDGET (window->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_hexand (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", window_weight_experiment,
02627         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     gtk_grid_attach (window->grid_experiment,
02636         GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02637     gtk_grid_attach (window->grid_experiment,
02638         GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02639     gtk_grid_attach (window->grid_experiment,
02640         GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02641     gtk_grid_attach (window->grid_experiment,
02642         GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02643     for (i = 0; i < MAX_NINPUTS; ++i)
02644     {
02645         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02646         window->check_template[i] = (GtkCheckButton *)
02647             gtk_check_button_new_with_label (buffer3);
02648         window->id_template[i]
02649             = g_signal_connect (window->check_template[i], "toggled",
02650                 window_inputs_experiment, NULL);
02651         gtk_grid_attach (window->grid_experiment,

```

```

02649         GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02650     window->button_template[i] = (GtkFileChooserButton *)
02651         gtk_file_chooser_button_new (_("Input template"),
02652             GTK_FILE_CHOOSER_ACTION_OPEN);
02653     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02654         _("Experimental input template file"));
02655     window->id_input[i] =
02656         g_signal_connect_swapped (window->button_template[i],
02657             "selection-changed",
02658             (GCallback) window_template_experiment,
02659             (void *) (size_t) i);
02660     gtk_grid_attach (window->grid_experiment,
02661         GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02662 }
02663 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02664 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02665     GTK_WIDGET (window->grid_experiment));
02666
02667 // Creating the error norm widgets
02668 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02669 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02670 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02671     GTK_WIDGET (window->grid_norm));
02672 window->button_norm[0] = (GtkRadioButton *)
02673     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02674 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02675     tip_norm[0]);
02676 gtk_grid_attach (window->grid_norm,
02677     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02678 g_signal_connect (window->button_norm[0], "clicked", 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);
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_hexpand (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
02705 // Creating the grid and attaching the widgets to the grid
02706 window->grid = (GtkGrid *) gtk_grid_new ();
02707 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02708 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02709 gtk_grid_attach (window->grid,
02710     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02711 gtk_grid_attach (window->grid,
02712     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02713 gtk_grid_attach (window->grid,
02714     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02715 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02716 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
02717     grid));
02718
02719 // Setting the window logo
02719 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02720 gtk_window_set_icon (window->window, window->logo);
02721
02722 // Showing the window
02723 gtk_widget_show_all (GTK_WIDGET (window->window));
02724
02725 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02726 #if GTK_MINOR_VERSION >= 16
02727     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02728     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02729     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02730     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02731     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02732     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02733     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);

```

```

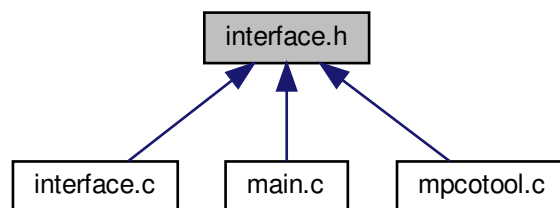
02734 #endif
02735
02736 // Reading initial example
02737 input_new ();
02738 buffer2 = g_get_current_dir ();
02739 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02740 g_free (buffer2);
02741 window_read (buffer);
02742 g_free (buffer);
02743
02744 #if DEBUG_INTERFACE
02745 fprintf (stderr, "window_new: start\n");
02746 #endif
02747 }

```

4.13 interface.h File Reference

Header file to define the graphical interface functions.

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



Data Structures

- struct [Options](#)
Struct to define the options dialog.
- struct [Running](#)
Struct to define the running dialog.
- struct [Window](#)
Struct to define the main window.

Macros

- #define [MAX_LENGTH](#) ([DEFAULT_PRECISION](#) + 8)
Max length of texts allowed in GtkSpinButtons.

Functions

- void [window_new](#) (GtkApplication *application)

Variables

- [Window window](#) [1]
Window struct to define the main interface window.

4.13.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.13.2 Function Documentation

4.13.2.1 window_new()

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

Function to open the main window.

Parameters

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

Definition at line [2066](#) of file [interface.c](#).

```
02067 {
02068     unsigned int i;
02069     char *buffer, *buffer2, buffer3[64];
02070     char *label_algorithm[NALGORITHMS] = {
02071         "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02072     };
02073     char *tip_algorithm[NALGORITHMS] = {
02074         _("Monte-Carlo brute force algorithm"),
02075         _("Sweep brute force algorithm"),
02076         _("Genetic algorithm"),
02077         _("Orthogonal sampling brute force algorithm"),
02078     };
02079     char *label_climbing[NCLIMBINGS] = {
02080         _("_Coordinates climbing"), _("_Random climbing")
02081     };
02082     char *tip_climbing[NCLIMBINGS] = {
02083         _("Coordinates climbing estimate method"),
```

```

02084     _("Random climbing estimate method")
02085 };
02086 char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02087 char *tip_norm[NNORMS] = {
02088     _("Euclidean error norm (L2)"),
02089     _("Maximum error norm (L)"),
02090     _("P error norm (Lp)"),
02091     _("Taxicab error norm (L1)")
02092 };
02093
02094 #if DEBUG_INTERFACE
02095     fprintf (stderr, "window_new: start\n");
02096 #endif
02097
02098     // Creating the window
02099     window->window = main_window
02100         = (GtkWindow *) gtk_application_window_new (application);
02101
02102     // Finish when closing the window
02103     g_signal_connect_swapped (window->window, "delete-event",
02104         G_CALLBACK (g_application_quit),
02105         G_APPLICATION (application));
02106
02107     // Setting the window title
02108     gtk_window_set_title (window->window, "MPCOTool");
02109
02110     // Creating the open button
02111     window->button_open = (GtkToolButton *) gtk_tool_button_new
02112         (gtk_image_new_from_icon_name ("document-open",
02113             GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02114     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02115
02116     // Creating the save button
02117     window->button_save = (GtkToolButton *) gtk_tool_button_new
02118         (gtk_image_new_from_icon_name ("document-save",
02119             GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02120     g_signal_connect (window->button_save, "clicked", (GCallback)
02121         window_save,
02122         NULL);
02123
02124     // Creating the run button
02125     window->button_run = (GtkToolButton *) gtk_tool_button_new
02126         (gtk_image_new_from_icon_name ("system-run",
02127             GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02128     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02129
02130     // Creating the options button
02131     window->button_options = (GtkToolButton *) gtk_tool_button_new
02132         (gtk_image_new_from_icon_name ("preferences-system",
02133             GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02134     g_signal_connect (window->button_options, "clicked",
02135         options_new, NULL);
02136
02137     // Creating the help button
02138     window->button_help = (GtkToolButton *) gtk_tool_button_new
02139         (gtk_image_new_from_icon_name ("help-browser",
02140             GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02141     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02142
02143     // Creating the about button
02144     window->button_about = (GtkToolButton *) gtk_tool_button_new
02145         (gtk_image_new_from_icon_name ("help-about",
02146             GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02147     g_signal_connect (window->button_about, "clicked",
02148         window_about, NULL);
02149
02150     // Creating the exit button
02151     window->button_exit = (GtkToolButton *) gtk_tool_button_new
02152         (gtk_image_new_from_icon_name ("application-exit",
02153             GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02154     g_signal_connect_swapped (window->button_exit, "clicked",
02155         G_CALLBACK (g_application_quit),
02156         G_APPLICATION (application));
02157
02158     // Creating the buttons bar
02159     window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02160     gtk_toolbar_insert
02161         (window->bar_buttons, GTK_TOOL_ITEM (window->
02162         button_open), 0);
02163     gtk_toolbar_insert
02164         (window->bar_buttons, GTK_TOOL_ITEM (window->
02165         button_save), 1);
02166     gtk_toolbar_insert
02167         (window->bar_buttons, GTK_TOOL_ITEM (window->
02168         button_run), 2);
02169     gtk_toolbar_insert
02170         (window->bar_buttons, GTK_TOOL_ITEM (window->

```

```

        button_options), 3);
02165     gtk_toolbar_insert
02166     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_help), 4);
02167     gtk_toolbar_insert
02168     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_about), 5);
02169     gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_exit), 6);
02171     gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02172
02173     // Creating the simulator program label and entry
02174     window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02175     window->button_simulator = (GtkFileChooserButton *)
02176     gtk_file_chooser_button_new (_("Simulator program"),
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",
02185     window_update, NULL);
02186     window->button_evaluator = (GtkFileChooserButton *)
02187     gtk_file_chooser_button_new (_("Evaluator program"),
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_hexexpand (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",
02238

```

```

window_update, NULL);
02239 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02240 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02241 window->spin_tolerance =
02242     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02243 gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->spin_tolerance),
02245      _("Tolerance to set the variable interval on the next iteration"));
02246 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02247 window->spin_bests
02248     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02249 gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->spin_bests),
02251      _("Number of best simulations used to set the variable interval "
02252        "on the next iteration"));
02253 window->label_population
02254     = (GtkLabel *) gtk_label_new (_("Population number"));
02255 window->spin_population
02256     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02257 gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_population),
02259      _("Number of population for the genetic algorithm"));
02260 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02261 window->label_generations
02262     = (GtkLabel *) gtk_label_new (_("Generations number"));
02263 window->spin_generations
02264     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02265 gtk_widget_set_tooltip_text
02266     (GTK_WIDGET (window->spin_generations),
02267      _("Number of generations for the genetic algorithm"));
02268 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02269 window->spin_mutation
02270     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02271 gtk_widget_set_tooltip_text
02272     (GTK_WIDGET (window->spin_mutation),
02273      _("Ratio of mutation for the genetic algorithm"));
02274 window->label_reproduction
02275     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02276 window->spin_reproduction
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02278 gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_reproduction),
02280      _("Ratio of reproduction for the genetic algorithm"));
02281 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02282 window->spin_adaptation
02283     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02284 gtk_widget_set_tooltip_text
02285     (GTK_WIDGET (window->spin_adaptation),
02286      _("Ratio of adaptation for the genetic algorithm"));
02287 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02288 window->spin_threshold = (GtkSpinButton *)
02289     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02290                                     precision[DEFAULT_PRECISION]);
02291 gtk_widget_set_tooltip_text
02292     (GTK_WIDGET (window->spin_threshold),
02293      _("Threshold in the objective function to finish the simulations"));
02294 window->scrolled_threshold =
02295     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02296 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02297     GTK_WIDGET (window->spin_threshold));
02298 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02299 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02300 //     GTK_ALIGN_FILL);
02301
02302 // Creating the hill climbing method properties
02303 window->check_climbing = (GtkCheckButton *)
02304     gtk_check_button_new_with_mnemonic (_("Hill climbing method"));
02305 g_signal_connect (window->check_climbing, "clicked",
02306     window_update, NULL);
02307 window->grid_climbing = (GtkGrid *) gtk_grid_new ();
02308 window->button_climbing[0] = (GtkRadioButton *)
02309     gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
02310 gtk_grid_attach (window->grid_climbing,
02311     GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
02312 g_signal_connect (window->button_climbing[0], "clicked",
02313     window_update, NULL);
02314 for (i = 0; ++i < NCLIMBINGS;)
02315 {
02316     window->button_climbing[i] = (GtkRadioButton *)
02317         gtk_radio_button_new_with_mnemonic
02318         (gtk_radio_button_get_group (window->button_climbing[0]),
02319          label_climbing[i]);
02320     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
02321         tip_climbing[i]);
02322     gtk_grid_attach (window->grid_climbing,
02323         GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
02324     g_signal_connect (window->button_climbing[i], "clicked",

```

```

        window_update,
02323         NULL);
02324     }
02325     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02326     window->spin_steps = (GtkSpinButton *)
02327         gtk_spin_button_new_with_range (1., 1.e12, 1.);
02328     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02329     window->label_estimates
02330         = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
02331     window->spin_estimates = (GtkSpinButton *)
02332         gtk_spin_button_new_with_range (1., 1.e3, 1.);
02333     window->label_relaxation
02334         = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02335     window->spin_relaxation = (GtkSpinButton *)
02336         gtk_spin_button_new_with_range (0., 2., 0.001);
02337     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_steps),
02338         0, NCLIMBINGS, 1, 1);
02339     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_steps),
02340         1, NCLIMBINGS, 1, 1);
02341     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_estimates),
02342         0, NCLIMBINGS + 1, 1, 1);
02343     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_estimates),
02344         1, NCLIMBINGS + 1, 1, 1);
02345     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_relaxation),
02346         0, NCLIMBINGS + 2, 1, 1);
02347     gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_relaxation),
02348         1, NCLIMBINGS + 2, 1, 1);
02349
02350     // Creating the array of algorithms
02351     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02352     window->button_algorithm[0] = (GtkRadioButton *)
02353         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02354     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02355         tip_algorithm[0]);
02356     gtk_grid_attach (window->grid_algorithm,
02357         GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02358     g_signal_connect (window->button_algorithm[0], "clicked",
02359         window_set_algorithm, NULL);
02360     for (i = 0; ++i < NALGORITHMS;)
02361     {
02362         window->button_algorithm[i] = (GtkRadioButton *)
02363             gtk_radio_button_new_with_mnemonic
02364             (gtk_radio_button_get_group (window->button_algorithm[0]),
02365             label_algorithm[i]);
02366         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02367             tip_algorithm[i]);
02368         gtk_grid_attach (window->grid_algorithm,
02369             GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02370         g_signal_connect (window->button_algorithm[i], "clicked",
02371             window_set_algorithm, NULL);
02372     }
02373     gtk_grid_attach (window->grid_algorithm,
02374         GTK_WIDGET (window->label_simulations),
02375         0, NALGORITHMS, 1, 1);
02376     gtk_grid_attach (window->grid_algorithm,
02377         GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02378     gtk_grid_attach (window->grid_algorithm,
02379         GTK_WIDGET (window->label_iterations),
02380         0, NALGORITHMS + 1, 1, 1);
02381     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_iterations),
02382         1, NALGORITHMS + 1, 1, 1);
02383     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_tolerance),
02384         0, NALGORITHMS + 2, 1, 1);
02385     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_tolerance),
02386         1, NALGORITHMS + 2, 1, 1);
02387     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_bests),
02388         0, NALGORITHMS + 3, 1, 1);
02389     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_bests),
02390         1, NALGORITHMS + 3, 1, 1);
02391     gtk_grid_attach (window->grid_algorithm,
02392         GTK_WIDGET (window->label_population),
02393         0, NALGORITHMS + 4, 1, 1);
02394     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_population),
02395         1, NALGORITHMS + 4, 1, 1);
02396     gtk_grid_attach (window->grid_algorithm,

```



```

02397         GTK_WIDGET (window->label_generations),
02398         0, NALGORITHMS + 5, 1, 1);
02399     gtk_grid_attach (window->grid_algorithm,
02400         GTK_WIDGET (window->spin_generations),
02401         1, NALGORITHMS + 5, 1, 1);
02402     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02403         window->label_mutation),
02404         0, NALGORITHMS + 6, 1, 1);
02405     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02406         window->spin_mutation),
02407         1, NALGORITHMS + 6, 1, 1);
02408     gtk_grid_attach (window->grid_algorithm,
02409         GTK_WIDGET (window->label_reproduction),
02410         0, NALGORITHMS + 7, 1, 1);
02411     gtk_grid_attach (window->grid_algorithm,
02412         GTK_WIDGET (window->spin_reproduction),
02413         1, NALGORITHMS + 7, 1, 1);
02414     gtk_grid_attach (window->grid_algorithm,
02415         GTK_WIDGET (window->label_adaptation),
02416         0, NALGORITHMS + 8, 1, 1);
02417     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02418         window->spin_adaptation),
02419         1, NALGORITHMS + 8, 1, 1);
02420     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02421         window->check_climbing),
02422         0, NALGORITHMS + 9, 2, 1);
02423     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02424         window->grid_climbing),
02425         0, NALGORITHMS + 10, 2, 1);
02426     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02427         window->label_threshold),
02428         0, NALGORITHMS + 11, 1, 1);
02429     gtk_grid_attach (window->grid_algorithm,
02430         GTK_WIDGET (window->scrolled_threshold),
02431         1, NALGORITHMS + 11, 1, 1);
02432     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02433     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02434         GTK_WIDGET (window->grid_algorithm));
02435
02436     // Creating the variable widgets
02437     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02438     gtk_widget_set_tooltip_text
02439         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02440     window->id_variable = g_signal_connect
02441         (window->combo_variable, "changed", window_set_variable, NULL);
02442     window->button_add_variable = (GtkButton *)
02443         gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
02444     g_signal_connect (window->button_add_variable, "clicked",
02445         window_add_variable,
02446         NULL);
02447     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
02448         _("Add variable"));
02449     window->button_remove_variable = (GtkButton *)
02450         gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
02451     g_signal_connect (window->button_remove_variable, "clicked",
02452         window_remove_variable, NULL);
02453     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
02454         _("Remove variable"));
02455     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02456     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02457     gtk_widget_set_tooltip_text
02458         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02459     gtk_widget_set_expand (GTK_WIDGET (window->entry_variable), TRUE);
02460     window->id_variable_label = g_signal_connect
02461         (window->entry_variable, "changed",
02462         window_label_variable, NULL);
02463     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02464     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02465         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02466     gtk_widget_set_tooltip_text
02467         (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02468     window->scrolled_min
02469         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02470     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02471         GTK_WIDGET (window->spin_min));
02472     g_signal_connect (window->spin_min, "value-changed",
02473         window_rangemin_variable, NULL);
02474     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02475     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02476         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02477     gtk_widget_set_tooltip_text
02478         (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02479     window->scrolled_max
02480         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02481     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02482         GTK_WIDGET (window->spin_max));
02483     g_signal_connect (window->spin_max, "value-changed",

```

```

02476         window_rangemax_variable, NULL);
02477 window->check_minabs = (GtkCheckButton *)
02478     gtk_check_button_new_with_mnemonic (_("Absolute minimum"));
02479 g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
02480 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02481     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02482 gtk_widget_set_tooltip_text
02483     (GTK_WIDGET (window->spin_minabs),
02484      _("Minimum allowed value of the variable"));
02485 window->scrolled_minabs
02486     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02487 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02488     GTK_WIDGET (window->spin_minabs));
02489 g_signal_connect (window->spin_minabs, "value-changed",
02490     window_rangeminabs_variable, NULL);
02491 window->check_maxabs = (GtkCheckButton *)
02492     gtk_check_button_new_with_mnemonic (_("Absolute maximum"));
02493 g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02494 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02495     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02496 gtk_widget_set_tooltip_text
02497     (GTK_WIDGET (window->spin_maxabs),
02498      _("Maximum allowed value of the variable"));
02499 window->scrolled_maxabs
02500     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02501 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02502     GTK_WIDGET (window->spin_maxabs));
02503 g_signal_connect (window->spin_maxabs, "value-changed",
02504     window_rangemaxabs_variable, NULL);
02505 window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02506 window->spin_precision = (GtkSpinButton *)
02507     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02508 gtk_widget_set_tooltip_text
02509     (GTK_WIDGET (window->spin_precision),
02510      _("Number of precision floating point digits\n"
02511        "0 is for integer numbers"));
02512 g_signal_connect (window->spin_precision, "value-changed",
02513     window_precision_variable, NULL);
02514 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02515 window->spin_sweeps =
02516     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02517 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02518     _("Number of steps sweeping the variable"));
02519 g_signal_connect (window->spin_sweeps, "value-changed",
02520     window_update_variable, NULL);
02521 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02522 window->spin_bits
02523     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02524 gtk_widget_set_tooltip_text
02525     (GTK_WIDGET (window->spin_bits),
02526      _("Number of bits to encode the variable"));
02527 g_signal_connect
02528     (window->spin_bits, "value-changed", window_update_variable, NULL);
;
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);
02558 gtk_grid_attach (window->grid_variable,
02559     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);

```

```

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

```

```

02642     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02643     window->check_template[i] = (GtkCheckButton *)
02644     gtk_check_button_new_with_label (buffer3);
02645     window->id_template[i]
02646     = g_signal_connect (window->check_template[i], "toggled",
02647     window_inputs_experiment, NULL);
02648     gtk_grid_attach (window->grid_experiment,
02649     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02650     window->button_template[i] = (GtkFileChooserButton *)
02651     gtk_file_chooser_button_new (_("Input template"),
02652     GTK_FILE_CHOOSER_ACTION_OPEN);
02653     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02654     _("Experimental input template file"));
02655     window->id_input[i] =
02656     g_signal_connect_swapped (window->button_template[i],
02657     "selection-changed",
02658     (GCallback) window_template_experiment,
02659     (void *) (size_t) i);
02660     gtk_grid_attach (window->grid_experiment,
02661     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02662 }
02663 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02664 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02665     GTK_WIDGET (window->grid_experiment));
02666
02667 // Creating the error norm widgets
02668 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02669 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02670 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02671     GTK_WIDGET (window->grid_norm));
02672 window->button_norm[0] = (GtkRadioButton *)
02673     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02674     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02675     tip_norm[0]);
02676     gtk_grid_attach (window->grid_norm,
02677     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02678     g_signal_connect (window->button_norm[0], "clicked",
02679     window_update, NULL);
02680     for (i = 0; ++i < NNORMS;)
02681     {
02682         window->button_norm[i] = (GtkRadioButton *)
02683         gtk_radio_button_new_with_mnemonic
02684         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02685         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02686         tip_norm[i]);
02687         gtk_grid_attach (window->grid_norm,
02688         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02689         g_signal_connect (window->button_norm[i], "clicked",
02690         window_update, NULL);
02691     }
02692     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02693     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02694     label_p), 1, 1, 1, 1);
02695     window->spin_p = (GtkSpinButton *)
02696     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02697     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02698     _("P parameter for the P error norm"));
02699     window->scrolled_p =
02700     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02701     gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02702     GTK_WIDGET (window->spin_p));
02703     gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02704     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02705     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02706     scrolled_p),
02707     1, 2, 1, 2);
02708
02709 // Creating the grid and attaching the widgets to the grid
02710 window->grid = (GtkGrid *) gtk_grid_new ();
02711     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02712     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02713     gtk_grid_attach (window->grid,
02714     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02715     gtk_grid_attach (window->grid,
02716     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02717     gtk_grid_attach (window->grid,
02718     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02719     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02720     gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
02721     window->grid));
02722
02723 // Setting the window logo
02724 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02725     gtk_window_set_icon (window->window, window->logo);
02726
02727 // Showing the window
02728     gtk_widget_show_all (GTK_WIDGET (window->window));

```

```

02724
02725 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02726 #if GTK_MINOR_VERSION >= 16
02727 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02728 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02729 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02730 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02731 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02732 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02733 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02734 #endif
02735
02736 // Reading initial example
02737 input_new ();
02738 buffer2 = g_get_current_dir ();
02739 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02740 g_free (buffer2);
02741 window_read (buffer);
02742 g_free (buffer);
02743
02744 #if DEBUG_INTERFACE
02745 fprintf (stderr, "window_new: start\n");
02746 #endif
02747 }

```

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 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 INTERFACE__H
00033 #define INTERFACE__H 1
00034
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
00048 typedef struct
00049 {
00050     GtkDialog *dialog;
00051     GtkGrid *grid;
00052     GtkLabel *label_seed;
00054     GtkSpinButton *spin_seed;
00056     GtkLabel *label_threads;
00057     GtkSpinButton *spin_threads;
00058     GtkLabel *label_climbing;
00059     GtkSpinButton *spin_climbing;
00060 } Options;
00061
00066 typedef struct
00067 {
00068     GtkDialog *dialog;
00069     GtkLabel *label;
00070     GtkSpinner *spinner;
00071     GtkGrid *grid;
00072 } Running;

```

```
00073
00078 typedef struct
00079 {
00080     GtkWidget *window;
00081     GtkGrid *grid;
00082     GtkToolBar *bar_buttons;
00083     GtkToolButton *button_open;
00084     GtkToolButton *button_save;
00085     GtkToolButton *button_run;
00086     GtkToolButton *button_options;
00087     GtkToolButton *button_help;
00088     GtkToolButton *button_about;
00089     GtkToolButton *button_exit;
00090     GtkGrid *grid_files;
00091     GtkLabel *label_simulator;
00092     GtkFileChooserButton *button_simulator;
00094     GtkCheckButton *check_evaluator;
00095     GtkFileChooserButton *button_evaluator;
00097     GtkLabel *label_result;
00098     GtkEntry *entry_result;
00099     GtkLabel *label_variables;
00100     GtkEntry *entry_variables;
00101     GtkFrame *frame_norm;
00102     GtkGrid *grid_norm;
00103     GtkRadioButton *button_norm[NNORMS];
00105     GtkLabel *label_p;
00106     GtkSpinButton *spin_p;
00107     GtkScrolledWindow *scrolled_p;
00109     GtkFrame *frame_algorithm;
00110     GtkGrid *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     GtkGrid *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     GtkGrid *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   GtkWidget *spin_step;
00184   GtkScrolledWindow *scrolled_step;
00185   GtkFrame *frame_experiment;
00186   GtkGrid *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 n_experiments;
00212   unsigned int n_variables;
00213 } Window;
00214
00215 // Global variables
00216 extern Window window[1];
00217
00218 // Public functions
00219 void window_new (GtkApplication * application);
00220
00221 #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 argn, char **argc)`

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
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
```



```

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

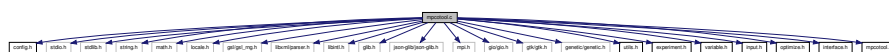
Main function source file.

```

#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"
#include "input.h"
#include "optimize.h"
#include "interface.h"
#include "mpcotool.h"

```

Include dependency graph for mpcotool.c:



Macros

- `#define DEBUG_MPCOTOOL 0`
Macro to debug main functions.

Functions

- int [mpcotool](#) (int argn, char **argc)

Variables

- GMutex [mutex](#) [1]
GMutex struct.
- int [ntasks](#)
Tasks number.
- unsigned int [nthreads](#)
Threads number.

4.17.1 Detailed Description

Main function source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.17.2 Function Documentation

4.17.2.1 mpcotool()

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

Main function.

Returns

0 on success, >0 on error.

Parameters

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

Definition at line 83 of file [mpcotool.c](#).

```

00095 {
00096 #if HAVE_GTK
00097     GtkApplication *application;
00098     char *buffer;
00099 #endif
00100
00101     // Starting pseudo-random numbers generator
00102 #if DEBUG_MPCOTOOL
00103     fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00104 #endif
00105     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00106
00107     // Allowing spaces in the XML data file
00108 #if DEBUG_MPCOTOOL
00109     fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00110 #endif
00111     xmlKeepBlanksDefault (0);
00112
00113     // Starting MPI
00114 #if HAVE_MPI
00115 #if DEBUG_MPCOTOOL
00116     fprintf (stderr, "mpcotool: starting MPI\n");
00117 #endif
00118     MPI_Init (&argn, &argc);
00119     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00120     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00121     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00122 #else
00123     ntasks = 1;
00124 #endif
00125
00126     // Resetting result and variables file names
00127 #if DEBUG_MPCOTOOL
00128     fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00129 #endif
00130     input->result = input->variables = NULL;
00131
00132     // Getting threads number and pseudo-random numbers generator seed
00133     nthreads_climbing = nthreads = cores_number ();
00134     optimize->seed = DEFAULT_RANDOM_SEED;
00135
00136 #if HAVE_GTK
00137
00138     // Setting local language and international floating point numbers notation
00139     setlocale (LC_ALL, "");
00140     setlocale (LC_NUMERIC, "C");
00141     window->application_directory = g_get_current_dir ();
00142     buffer = g_build_filename (window->application_directory,
00143                               LOCALE_DIR, NULL);
00144     bindtextdomain (PROGRAM_INTERFACE, buffer);
00145     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00146     textdomain (PROGRAM_INTERFACE);
00147
00148     // Initing GTK+
00149     gtk_disable_setlocale ();
00150     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00151                                       G_APPLICATION_FLAGS_NONE);
00152     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00153
00154     // Opening the main window
00155     g_application_run (G_APPLICATION (application), 0, NULL);
00156
00157     // Freeing memory
00158     input_free ();
00159     g_free (buffer);
00160     gtk_widget_destroy (GTK_WIDGET (window->window));
00161     g_object_unref (application);
00162     g_free (window->application_directory);
00163 #else
00164
00165     // Checking syntax
00166     if (argn < 2)
00167     {
00168         printf ("The syntax is:\n"
00169                "    ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00170                "[variables_file]\n");
00171         return 1;
00172     }
00173
00174     // Getting threads number and pseudo-random numbers generator seed
00175 #if DEBUG_MPCOTOOL
00176     fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
00177             "generator seed\n");

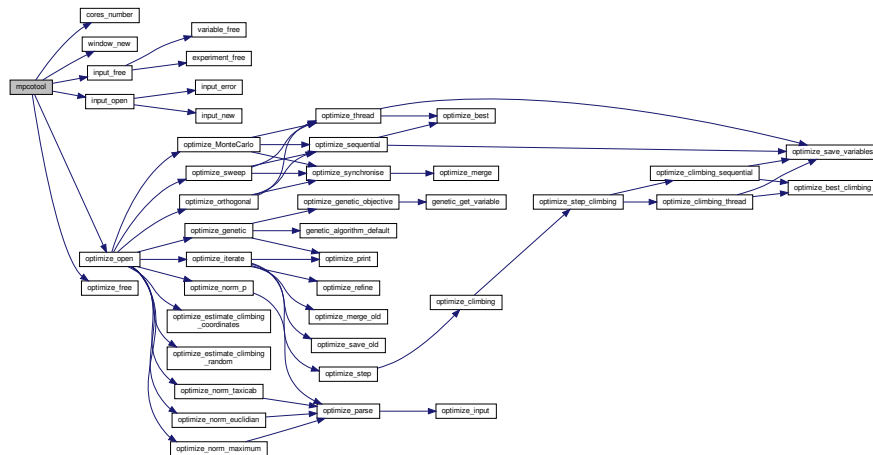
```

```

00178 #endif
00179 if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00180 {
00181     nthreads_climbing = nthreads = atoi (argc[2]);
00182     if (!nthreads)
00183     {
00184         printf ("Bad threads number\n");
00185         return 2;
00186     }
00187     argc += 2;
00188     argn -= 2;
00189     if (argn > 2 && !strcmp (argc[1], "-seed"))
00190     {
00191         optimize->seed = atoi (argc[2]);
00192         argc += 2;
00193         argn -= 2;
00194     }
00195 }
00196 else if (argn > 2 && !strcmp (argc[1], "-seed"))
00197 {
00198     optimize->seed = atoi (argc[2]);
00199     argc += 2;
00200     argn -= 2;
00201     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00202     {
00203         nthreads_climbing = nthreads = atoi (argc[2]);
00204         if (!nthreads)
00205         {
00206             printf ("Bad threads number\n");
00207             return 2;
00208         }
00209         argc += 2;
00210         argn -= 2;
00211     }
00212 }
00213 printf ("nthreads=%u\n", nthreads);
00214 printf ("seed=%lu\n", optimize->seed);
00215
00216 // Checking arguments
00217 #if DEBUG_MPCOTOOL
00218 fprintf (stderr, "mpcotool: checking arguments\n");
00219 #endif
00220 if (argn > 4 || argn < 2)
00221 {
00222     printf ("The syntax is:\n"
00223            "    ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00224            "[variables_file]\n");
00225     return 1;
00226 }
00227 if (argn > 2)
00228     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00229 if (argn == 4)
00230     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00231
00232 // Making optimization
00233 #if DEBUG_MPCOTOOL
00234 fprintf (stderr, "mpcotool: making optimization\n");
00235 #endif
00236 if (input_open (argc[1]))
00237     optimize_open ();
00238
00239 // Freeing memory
00240 #if DEBUG_MPCOTOOL
00241 fprintf (stderr, "mpcotool: freeing memory and closing\n");
00242 #endif
00243 optimize_free ();
00244
00245 #endif
00246
00247 // Closing MPI
00248 #if HAVE_MPI
00249 MPI_Finalize ();
00250 #endif
00251
00252 // Freeing memory
00253 gsl_rng_free (optimize->rng);
00254
00255 // Closing
00256 return 0;
00257 }

```

Here is the call graph for this function:



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>
  
```

```

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 #define DEBUG_MPCOTOOL 0
00072
00073 GMutex mutex[1];
00074 int ntasks;
00075 unsigned int nthreads;
00076
00082 int
00083 mpcotool (int argn
00084 #if HAVE_GTK
00085     __attribute__ ((unused))
00086 #endif
00087     ,
00089     char **argc
00090 #if HAVE_GTK
00091     __attribute__ ((unused))
00092 #endif
00093 )
00094 {
00095     {
00096         #if HAVE_GTK
00097             GtkApplication *application;
00098             char *buffer;
00099         #endif
00100
00101         // Starting pseudo-random numbers generator
00102         #if DEBUG_MPCOTOOL
00103             fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00104         #endif
00105         optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00106
00107         // Allowing spaces in the XML data file
00108         #if DEBUG_MPCOTOOL
00109             fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00110         #endif
00111         xmlKeepBlanksDefault (0);
00112
00113         // Starting MPI
00114         #if HAVE_MPI
00115         #if DEBUG_MPCOTOOL
00116             fprintf (stderr, "mpcotool: starting MPI\n");
00117         #endif
00118             MPI_Init (&argn, &argc);
00119             MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00120             MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00121             printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00122         #else
00123             ntasks = 1;
00124         #endif
00125
00126         // Resetting result and variables file names
00127         #if DEBUG_MPCOTOOL
00128             fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00129         #endif
00130         input->result = input->variables = NULL;
00131
00132         // Getting threads number and pseudo-random numbers generator seed
00133         nthreads_climbing = nthreads = cores_number ();
00134         optimize->seed = DEFAULT_RANDOM_SEED;
00135
00136         #if HAVE_GTK
00137
00138         // Setting local language and international floating point numbers notation
00139         setlocale (LC_ALL, "");
00140         setlocale (LC_NUMERIC, "C");
00141         window->application_directory = g_get_current_dir ();
00142         buffer = g_build_filename (window->application_directory,
00143             LOCALE_DIR, NULL);
00143         bindtextdomain (PROGRAM_INTERFACE, buffer);
00144         bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00145         textdomain (PROGRAM_INTERFACE);
00146
00147         // Initing GTK+
00148         gtk_disable_setlocale ();
00149         application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00150             G_APPLICATION_FLAGS_NONE);

```

```

00151 g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00152
00153 // Opening the main window
00154 g_application_run (G_APPLICATION (application), 0, NULL);
00155
00156 // Freeing memory
00157 input_free ();
00158 g_free (buffer);
00159 gtk_widget_destroy (GTK_WIDGET (window->window));
00160 g_object_unref (application);
00161 g_free (window->application_directory);
00162
00163 #else
00164
00165 // Checking syntax
00166 if (argn < 2)
00167 {
00168     printf ("The syntax is:\n"
00169             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00170             "[variables_file]\n");
00171     return 1;
00172 }
00173
00174 // Getting threads number and pseudo-random numbers generator seed
00175 #if DEBUG_MPCOTOOL
00176 fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
00177         "generator seed\n");
00178 #endif
00179 if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00180 {
00181     nthreads_climbing = nthreads = atoi (argc[2]);
00182     if (!nthreads)
00183     {
00184         printf ("Bad threads number\n");
00185         return 2;
00186     }
00187     argc += 2;
00188     argn -= 2;
00189     if (argn > 2 && !strcmp (argc[1], "-seed"))
00190     {
00191         optimize->seed = atoi (argc[2]);
00192         argc += 2;
00193         argn -= 2;
00194     }
00195 }
00196 else if (argn > 2 && !strcmp (argc[1], "-seed"))
00197 {
00198     optimize->seed = atoi (argc[2]);
00199     argc += 2;
00200     argn -= 2;
00201     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00202     {
00203         nthreads_climbing = nthreads = atoi (argc[2]);
00204         if (!nthreads)
00205         {
00206             printf ("Bad threads number\n");
00207             return 2;
00208         }
00209         argc += 2;
00210         argn -= 2;
00211     }
00212 }
00213 printf ("nthreads=%u\n", nthreads);
00214 printf ("seed=%lu\n", optimize->seed);
00215
00216 // Checking arguments
00217 #if DEBUG_MPCOTOOL
00218 fprintf (stderr, "mpcotool: checking arguments\n");
00219 #endif
00220 if (argn > 4 || argn < 2)
00221 {
00222     printf ("The syntax is:\n"
00223             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00224             "[variables_file]\n");
00225     return 1;
00226 }
00227 if (argn > 2)
00228     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00229 if (argn == 4)
00230     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00231
00232 // Making optimization
00233 #if DEBUG_MPCOTOOL
00234 fprintf (stderr, "mpcotool: making optimization\n");
00235 #endif
00236 if (input_open (argc[1]))
00237     optimize_open ();

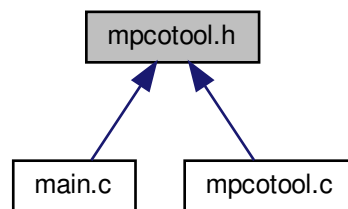
```

```
00238
00239 // Freeing memory
00240 #if DEBUG_MPCOTOOL
00241     fprintf (stderr, "mpcotool: freeing memory and closing\n");
00242 #endif
00243     optimize_free ();
00244 #endif
00245
00246 // Closing MPI
00247 #if HAVE_MPI
00248     MPI_Finalize ();
00249 #endif
00250
00251 // Freeing memory
00252     gsl_rng_free (optimize->rng);
00253
00254 // Closing
00255     return 0;
00256 }
00257 }
```

4.19 mpcotool.h File Reference

Main function header file.

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



Functions

- int [mpcotool](#) (int argn, char **argc)

4.19.1 Detailed Description

Main function header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.19.2 Function Documentation

4.19.2.1 mpcotool()

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

Main function.

Returns

0 on success, >0 on error.

Parameters

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

Definition at line 83 of file [mpcotool.c](#).

```
00095 {
00096     #if HAVE_GTK
00097         GtkApplication *application;
00098         char *buffer;
00099     #endif
00100
00101     // Starting pseudo-random numbers generator
00102     #if DEBUG_MPCOTOOL
00103         fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
00104     #endif
00105     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00106
00107     // Allowing spaces in the XML data file
00108     #if DEBUG_MPCOTOOL
00109         fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
00110     #endif
00111     xmlKeepBlanksDefault (0);
00112
00113     // Starting MPI
00114     #if HAVE_MPI
00115     #if DEBUG_MPCOTOOL
00116         fprintf (stderr, "mpcotool: starting MPI\n");
00117     #endif
00118     MPI_Init (&argn, &argc);
00119     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00120     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00121     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00122     #else
00123         ntasks = 1;
00124     #endif
00125
00126     // Resetting result and variables file names
00127     #if DEBUG_MPCOTOOL
00128         fprintf (stderr, "mpcotool: resetting result and variables file names\n");
00129     #endif
00130     input->result = input->variables = NULL;
00131
00132     // Getting threads number and pseudo-random numbers generator seed
00133     nthreads_climbing = nthreads = cores_number ();
00134     optimize->seed = DEFAULT_RANDOM_SEED;
00135
00136     #if HAVE_GTK
00137
00138     // Setting local language and international floating point numbers notation
```

```

00139  setlocale (LC_ALL, "");
00140  setlocale (LC_NUMERIC, "C");
00141  window->application_directory = g_get_current_dir ();
00142  buffer = g_build_filename (window->application_directory,
    LOCALE_DIR, NULL);
00143  bindtextdomain (PROGRAM_INTERFACE, buffer);
00144  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00145  textdomain (PROGRAM_INTERFACE);
00146
00147  // Initing GTK+
00148  gtk_disable_setlocale ();
00149  application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
    G_APPLICATION_FLAGS_NONE);
00150  g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00151
00152  // Opening the main window
00153  g_application_run (G_APPLICATION (application), 0, NULL);
00154
00155  // Freeing memory
00156  input_free ();
00157  g_free (buffer);
00158  gtk_widget_destroy (GTK_WIDGET (window->window));
00159  g_object_unref (application);
00160  g_free (window->application_directory);
00161
00162  #else
00163
00164  // Checking syntax
00165  if (argn < 2)
00166  {
00167      printf ("The syntax is:\n"
00168              "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00169              "[variables_file]\n");
00170      return 1;
00171  }
00172
00173  // Getting threads number and pseudo-random numbers generator seed
00174  #if DEBUG_MPCOTOOL
00175  fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
00176           "generator seed\n");
00177  #endif
00178  if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00179  {
00180      nthreads_climbing = nthreads = atoi (argc[2]);
00181      if (!nthreads)
00182      {
00183          printf ("Bad threads number\n");
00184          return 2;
00185      }
00186      argc += 2;
00187      argn -= 2;
00188      if (argn > 2 && !strcmp (argc[1], "-seed"))
00189      {
00190          optimize->seed = atoi (argc[2]);
00191          argc += 2;
00192          argn -= 2;
00193      }
00194  }
00195  else if (argn > 2 && !strcmp (argc[1], "-seed"))
00196  {
00197      optimize->seed = atoi (argc[2]);
00198      argc += 2;
00199      argn -= 2;
00200      if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00201      {
00202          nthreads_climbing = nthreads = atoi (argc[2]);
00203          if (!nthreads)
00204          {
00205              printf ("Bad threads number\n");
00206              return 2;
00207          }
00208          argc += 2;
00209          argn -= 2;
00210      }
00211  }
00212  printf ("nthreads=%u\n", nthreads);
00213  printf ("seed=%lu\n", optimize->seed);
00214
00215  // Checking arguments
00216  #if DEBUG_MPCOTOOL
00217  fprintf (stderr, "mpcotool: checking arguments\n");
00218  #endif
00219  if (argn > 4 || argn < 2)
00220  {
00221      printf ("The syntax is:\n"
00222              "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00223              "[variables_file]\n");
00224  }

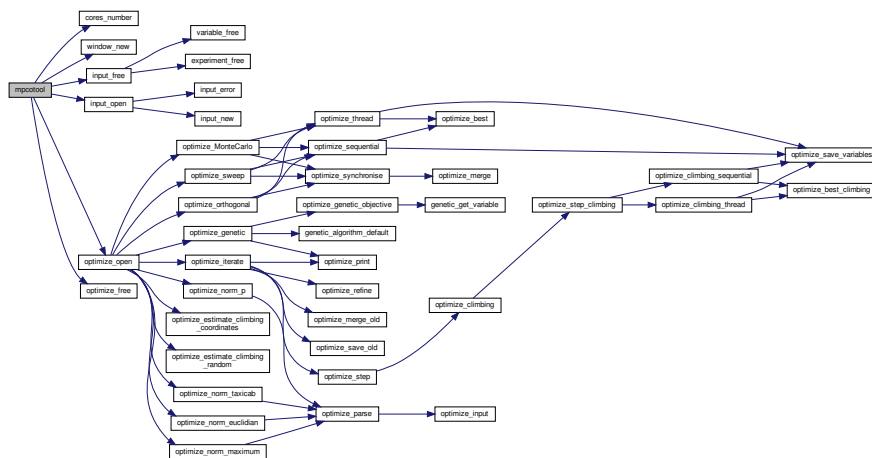
```

```

00225     return 1;
00226 }
00227 if (argn > 2)
00228     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00229 if (argn == 4)
00230     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00231
00232 // Making optimization
00233 #if DEBUG_MPCOTOOL
00234 fprintf (stderr, "mpcotool: making optimization\n");
00235 #endif
00236 if (input_open (argc[1]))
00237     optimize_open ();
00238
00239 // Freeing memory
00240 #if DEBUG_MPCOTOOL
00241 fprintf (stderr, "mpcotool: freeing memory and closing\n");
00242 #endif
00243 optimize_free ();
00244
00245 #endif
00246
00247 // Closing MPI
00248 #if HAVE_MPI
00249 MPI_Finalize ();
00250 #endif
00251
00252 // Freeing memory
00253 gsl_rng_free (optimize->rng);
00254
00255 // Closing
00256 return 0;
00257 }

```

Here is the call graph for this function:



4.20 mpcotool.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 MPCOTOOL__H
00033 #define MPCOTOOL__H 1
00034
00035 extern int mpcotool (int argn, char **argc);
00036
00037 #endif

```

4.21 optimize.c File Reference

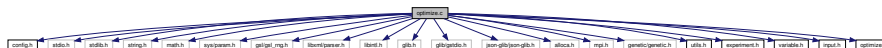
Source file to define the optimization functions.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/param.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <alloca.h>
#include <mpi.h>
#include "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

- static void [optimize_input](#) (unsigned int simulation, char *[input](#), GMappedFile *[stencil](#))
- static double [optimize_parse](#) (unsigned int simulation, unsigned int experiment)
- static double [optimize_norm_euclidian](#) (unsigned int simulation)
- static double [optimize_norm_maximum](#) (unsigned int simulation)
- static double [optimize_norm_p](#) (unsigned int simulation)
- static double [optimize_norm_taxicab](#) (unsigned int simulation)
- static void [optimize_print](#) ()
- static void [optimize_save_variables](#) (unsigned int simulation, double error)
- static void [optimize_best](#) (unsigned int simulation, double value)
- static void [optimize_sequential](#) ()
- static void * [optimize_thread](#) (ParallelData *data)
- static void [optimize_merge](#) (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)
- static void [optimize_synchronise](#) ()
- static void [optimize_sweep](#) ()
- static void [optimize_MonteCarlo](#) ()
- static void [optimize_orthogonal](#) ()
- static void [optimize_best_climbing](#) (unsigned int simulation, double value)
- static void [optimize_climbing_sequential](#) (unsigned int simulation)
- static void * [optimize_climbing_thread](#) (ParallelData *data)
- static double [optimize_estimate_climbing_random](#) (unsigned int variable, unsigned int estimate)
- static double [optimize_estimate_climbing_coordinates](#) (unsigned int variable, unsigned int estimate)
- static void [optimize_step_climbing](#) (unsigned int simulation)
- static void [optimize_climbing](#) ()
- static double [optimize_genetic_objective](#) (**Entity** *entity)
- static void [optimize_genetic](#) ()
- static void [optimize_save_old](#) ()
- static void [optimize_merge_old](#) ()
- static void [optimize_refine](#) ()
- static void [optimize_step](#) ()
- static void [optimize_iterate](#) ()
- void [optimize_free](#) ()
- void [optimize_open](#) ()

Variables

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

4.21.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.21.2 Function Documentation

4.21.2.1 optimize_best()

```
static void optimize_best (
    unsigned int simulation,
    double value ) [static]
```

Function to save the best simulations.

Parameters

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

Definition at line [449](#) of file [optimize.c](#).

```
00451 {
00452     unsigned int i, j;
00453     double e;
00454     #if DEBUG_OPTIMIZE
00455     fprintf (stderr, "optimize_best: start\n");
00456     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00457             optimize->nsaveds, optimize->nbest);
00458     #endif
00459     if (optimize->nsaveds < optimize->nbest
00460         || value < optimize->error_best[optimize->nsaveds - 1])
00461     {
00462         if (optimize->nsaveds < optimize->nbest)
00463             ++optimize->nsaveds;
00464         optimize->error_best[optimize->nsaveds - 1] = value;
00465         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00466         for (i = optimize->nsaveds; --i;)
00467         {
00468             if (optimize->error_best[i] < optimize->
00469                 error_best[i - 1])
00470             {
00471                 j = optimize->simulation_best[i];
00472                 e = optimize->error_best[i];
00473                 optimize->simulation_best[i] = optimize->
00474                     simulation_best[i - 1];
```

```

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

```

4.21.2.2 optimize_best_climbing()

```

static void optimize_best_climbing (
    unsigned int simulation,
    double value ) [static]

```

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

Parameters

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

Definition at line 811 of file [optimize.c](#).

```

00813 {
00814 #if DEBUG_OPTIMIZE
00815     fprintf(stderr, "optimize_best_climbing: start\n");
00816     fprintf(stderr,
00817         "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00818         simulation, value, optimize->error_best[0]);
00819 #endif
00820     if (value < optimize->error_best[0])
00821     {
00822         optimize->error_best[0] = value;
00823         optimize->simulation_best[0] = simulation;
00824 #if DEBUG_OPTIMIZE
00825         fprintf(stderr,
00826             "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00827             simulation, value);
00828 #endif
00829     }
00830 #if DEBUG_OPTIMIZE
00831     fprintf(stderr, "optimize_best_climbing: end\n");
00832 #endif
00833 }

```

4.21.2.3 optimize_climbing()

```

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

```

Function to optimize with a hill climbing method.

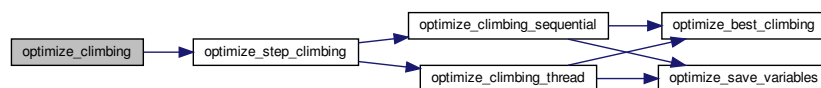
Definition at line 1039 of file [optimize.c](#).

```

01040 {
01041     unsigned int i, j, k, b, s, adjust;
01042     #if DEBUG_OPTIMIZE
01043         fprintf (stderr, "optimize_climbing: start\n");
01044     #endif
01045     for (i = 0; i < optimize->nvariables; ++i)
01046         optimize->climbing[i] = 0.;
01047     b = optimize->simulation_best[0] * optimize->
nvariables;
01048     s = optimize->nsimulations;
01049     adjust = 1;
01050     for (i = 0; i < optimize->nsteps; ++i, s += optimize->
nestimates, b = k)
01051     {
01052         #if DEBUG_OPTIMIZE
01053             fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01054                     i, optimize->simulation_best[0]);
01055         #endif
01056         optimize_step_climbing (s);
01057         k = optimize->simulation_best[0] * optimize->
nvariables;
01058         #if DEBUG_OPTIMIZE
01059             fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01060                     i, optimize->simulation_best[0]);
01061         #endif
01062         if (k == b)
01063         {
01064             if (adjust)
01065                 for (j = 0; j < optimize->nvariables; ++j)
01066                     optimize->step[j] *= 0.5;
01067             for (j = 0; j < optimize->nvariables; ++j)
01068                 optimize->climbing[j] = 0.;
01069             adjust = 1;
01070         }
01071         else
01072         {
01073             for (j = 0; j < optimize->nvariables; ++j)
01074             {
01075                 #if DEBUG_OPTIMIZE
01076                     fprintf (stderr,
01077                             "optimize_climbing: best%u=%.14le old%u=%.14le\n",
01078                             j, optimize->value[k + j], j, optimize->
value[b + j]);
01079                 #endif
01080                 optimize->climbing[j]
01081                     = (1. - optimize->relaxation) * optimize->
climbing[j]
01082                     + optimize->relaxation
01083                     * (optimize->value[k + j] - optimize->value[b + j]);
01084                 #if DEBUG_OPTIMIZE
01085                     fprintf (stderr, "optimize_climbing: climbing%u=%.14le\n",
01086                             j, optimize->climbing[j]);
01087                 #endif
01088             }
01089             adjust = 0;
01090         }
01091     }
01092     #if DEBUG_OPTIMIZE
01093         fprintf (stderr, "optimize_climbing: end\n");
01094     #endif
01095 }

```

Here is the call graph for this function:



4.21.2.4 optimize_climbing_sequential()

```

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

```


Function to estimate the hill climbing sequentially.

Parameters

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

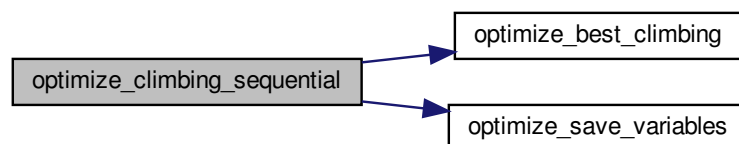
Definition at line 839 of file [optimize.c](#).

```

00840 {
00841     double e;
00842     unsigned int i, j;
00843     #if DEBUG_OPTIMIZE
00844     fprintf (stderr, "optimize_climbing_sequential: start\n");
00845     fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00846             "nend_climbing=%u\n",
00847             optimize->nstart_climbing, optimize->
nend_climbing);
00848     #endif
00849     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00850     {
00851         j = simulation + i;
00852         e = optimize_norm (j);
00853         optimize_best_climbing (j, e);
00854         optimize_save_variables (j, e);
00855         if (e < optimize->threshold)
00856         {
00857             optimize->stop = 1;
00858             break;
00859         }
00860     #if DEBUG_OPTIMIZE
00861     fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
00862     #endif
00863     }
00864     #if DEBUG_OPTIMIZE
00865     fprintf (stderr, "optimize_climbing_sequential: end\n");
00866     #endif
00867 }

```

Here is the call graph for this function:



4.21.2.5 optimize_climbing_thread()

```

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

```

Function to estimate the hill climbing on a thread.

Returns

NULL

Parameters

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

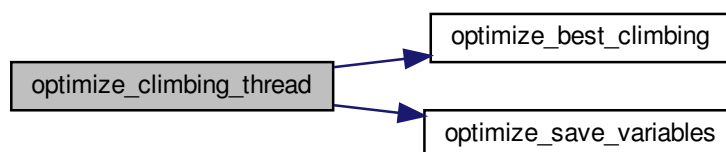
Definition at line 875 of file [optimize.c](#).

```

00876 {
00877     unsigned int i, thread;
00878     double e;
00879     #if DEBUG_OPTIMIZE
00880     fprintf (stderr, "optimize_climbing_thread: start\n");
00881     #endif
00882     thread = data->thread;
00883     #if DEBUG_OPTIMIZE
00884     fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00885             thread,
00886             optimize->thread_climbing[thread],
00887             optimize->thread_climbing[thread + 1]);
00888     #endif
00889     for (i = optimize->thread_climbing[thread];
00890          i < optimize->thread_climbing[thread + 1]; ++i)
00891     {
00892         e = optimize_norm (i);
00893         g_mutex_lock (mutex);
00894         optimize_best_climbing (i, e);
00895         optimize_save_variables (i, e);
00896         if (e < optimize->threshold)
00897             optimize->stop = 1;
00898         g_mutex_unlock (mutex);
00899         if (optimize->stop)
00900             break;
00901     #if DEBUG_OPTIMIZE
00902     fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00903     #endif
00904     }
00905     #if DEBUG_OPTIMIZE
00906     fprintf (stderr, "optimize_climbing_thread: end\n");
00907     #endif
00908     g_thread_exit (NULL);
00909     return NULL;
00910 }

```

Here is the call graph for this function:



4.21.2.6 optimize_estimate_climbing_coordinates()

```

static double optimize_estimate_climbing_coordinates (
    unsigned int variable,
    unsigned int estimate ) [static]

```

Function to estimate a component of the hill climbing vector.

Parameters

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

Definition at line 940 of file [optimize.c](#).

```

00944 {
00945     double x;
00946     #if DEBUG_OPTIMIZE
00947     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00948     #endif
00949     x = optimize->climbing[variable];
00950     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00951     {
00952         if (estimate & 1)
00953             x += optimize->step[variable];
00954         else
00955             x -= optimize->step[variable];
00956     }
00957     #if DEBUG_OPTIMIZE
00958     fprintf (stderr,
00959             "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
00960             variable, x);
00961     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00962     #endif
00963     return x;
00964 }
```

4.21.2.7 optimize_estimate_climbing_random()

```

static double optimize_estimate_climbing_random (
    unsigned int variable,
    unsigned int estimate ) [static]
```

Function to estimate a component of the hill climbing vector.

Parameters

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

Definition at line 916 of file [optimize.c](#).

```

00921 {
00922     double x;
00923     #if DEBUG_OPTIMIZE
00924     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00925     #endif
00926     x = optimize->climbing[variable]
00927         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00928         step[variable];
00929     #if DEBUG_OPTIMIZE
00929     fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
00930             variable, x);
00931     fprintf (stderr, "optimize_estimate_climbing_random: end\n");
00932     #endif
00933     return x;
00934 }
```

4.21.2.8 optimize_free()

```
void optimize_free ( )
```

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

Definition at line 1396 of file [optimize.c](#).

```
01397 {
01398     unsigned int i, j;
01399     #if DEBUG_OPTIMIZE
01400     fprintf (stderr, "optimize_free: start\n");
01401     #endif
01402     for (j = 0; j < optimize->ninputs; ++j)
01403     {
01404         for (i = 0; i < optimize->nexperiments; ++i)
01405             g_mapped_file_unref (optimize->file[j][i]);
01406         g_free (optimize->file[j]);
01407     }
01408     g_free (optimize->error_old);
01409     g_free (optimize->value_old);
01410     g_free (optimize->value);
01411     g_free (optimize->genetic_variable);
01412     #if DEBUG_OPTIMIZE
01413     fprintf (stderr, "optimize_free: end\n");
01414     #endif
01415 }
```

4.21.2.9 optimize_genetic()

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

Function to optimize with the genetic algorithm.

Definition at line 1136 of file [optimize.c](#).

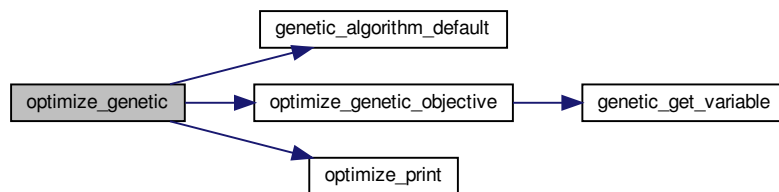
```
01137 {
01138     double *best_variable = NULL;
01139     char *best_genome = NULL;
01140     double best_objective = 0.;
01141     #if DEBUG_OPTIMIZE
01142     fprintf (stderr, "optimize_genetic: start\n");
01143     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01144   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

```

01169     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01170     optimize->error_old[0] = best_objective;
01171     memcpy (optimize->value_old, best_variable,
01172            optimize->nvariables * sizeof (double));
01173     g_free (best_genome);
01174     g_free (best_variable);
01175     optimize_print ();
01176     #if DEBUG_OPTIMIZE
01177     fprintf (stderr, "optimize_genetic: end\n");
01178     #endif
01179 }

```

Here is the call graph for this function:



4.21.2.10 optimize_genetic_objective()

```

static double optimize_genetic_objective (
    Entity * entity ) [static]

```

Function to calculate the objective function of an entity.

Returns

objective function value.

Parameters

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

Definition at line 1103 of file `optimize.c`.

```

01104 {
01105     unsigned int j;
01106     double objective;
01107     char buffer[64];
01108     #if DEBUG_OPTIMIZE
01109     fprintf (stderr, "optimize_genetic_objective: start\n");
01110     #endif
01111     for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113         optimize->value[entity->id * optimize->nvariables + j]
01114             = genetic_get_variable (entity, optimize->genetic_variable + j);
01115     }
01116     objective = optimize_norm (entity->id);

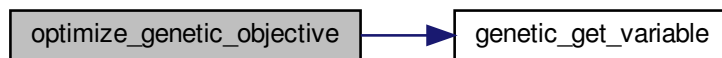
```

```

01117     g_mutex_lock (mutex);
01118     for (j = 0; j < optimize->nvariables; ++j)
01119     {
01120         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01121         fprintf (optimize->file_variables, buffer,
01122                 genetic_get_variable (entity, optimize->genetic_variable + j));
01123     }
01124     fprintf (optimize->file_variables, "%.14le\n", objective);
01125     g_mutex_unlock (mutex);
01126     #if DEBUG_OPTIMIZE
01127     fprintf (stderr, "optimize_genetic_objective: end\n");
01128     #endif
01129     return objective;
01130 }

```

Here is the call graph for this function:



4.21.2.11 optimize_input()

```

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

```

Function to write the simulation input file.

Parameters

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

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

```

00097 {
00098     char buffer[32], value[32];
00099     GRegex *regex;
00100     FILE *file;
00101     char *buffer2, *buffer3 = NULL, *content;
00102     gsize length;
00103     unsigned int i;
00104
00105     #if DEBUG_OPTIMIZE
00106     fprintf (stderr, "optimize_input: start\n");
00107     #endif
00108
00109     // Checking the file
00110     if (!stencil)
00111         goto optimize_input_end;

```

```

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

```

4.21.2.12 optimize_iterate()

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

Function to iterate the algorithm.

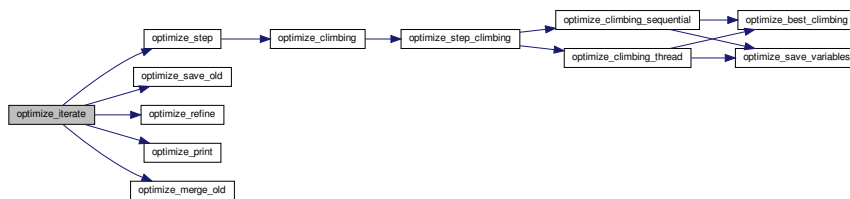
Definition at line 1366 of file [optimize.c](#).

```

01367 {
01368     unsigned int i;
01369     #if DEBUG_OPTIMIZE
01370     fprintf (stderr, "optimize_iterate: start\n");
01371     #endif
01372     optimize->error_old = (double *) g_malloc (optimize->
nbest * sizeof (double));
01373     optimize->value_old =
01374         (double *) g_malloc (optimize->nbest * optimize->
nvariables *
01375                               sizeof (double));
01376     optimize_step ();
01377     optimize_save_old ();
01378     optimize_refine ();
01379     optimize_print ();
01380     for (i = 1; i < optimize->niterations && !optimize->
stop; ++i)
01381     {
01382         optimize_step ();
01383         optimize_merge_old ();
01384         optimize_refine ();
01385         optimize_print ();
01386     }
01387     #if DEBUG_OPTIMIZE
01388     fprintf (stderr, "optimize_iterate: end\n");
01389     #endif
01390 }

```

Here is the call graph for this function:



4.21.2.13 optimize_merge()

```

static void optimize_merge (
    unsigned int nsaveds,
    unsigned int * simulation_best,
    double * error_best ) [inline], [static]

```

Function to merge the 2 optimization results.

Parameters

<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 562 of file [optimize.c](#).

```

00567 {
00568     unsigned int i, j, k, s[optimize->nbest];

```



```

00569     double e[optimize->nbest];
00570     #if DEBUG_OPTIMIZE
00571     fprintf (stderr, "optimize_merge: start\n");
00572     #endif
00573     i = j = k = 0;
00574     do
00575     {
00576         if (i == optimize->nsaveds)
00577         {
00578             s[k] = simulation_best[j];
00579             e[k] = error_best[j];
00580             ++j;
00581             ++k;
00582             if (j == nsaveds)
00583                 break;
00584         }
00585         else if (j == nsaveds)
00586         {
00587             s[k] = optimize->simulation_best[i];
00588             e[k] = optimize->error_best[i];
00589             ++i;
00590             ++k;
00591             if (i == optimize->nsaveds)
00592                 break;
00593         }
00594         else if (optimize->error_best[i] > error_best[j])
00595         {
00596             s[k] = simulation_best[j];
00597             e[k] = error_best[j];
00598             ++j;
00599             ++k;
00600         }
00601         else
00602         {
00603             s[k] = optimize->simulation_best[i];
00604             e[k] = optimize->error_best[i];
00605             ++i;
00606             ++k;
00607         }
00608     }
00609     while (k < optimize->nbest);
00610     optimize->nsaveds = k;
00611     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00612     memcpy (optimize->error_best, e, k * sizeof (double));
00613     #if DEBUG_OPTIMIZE
00614     fprintf (stderr, "optimize_merge: end\n");
00615     #endif
00616 }

```

4.21.2.14 optimize_merge_old()

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

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

Definition at line 1217 of file [optimize.c](#).

```

01218 {
01219     unsigned int i, j, k;
01220     double v[optimize->nbest * optimize->nvariables], e[
optimize->nbest],
01221     *enew, *eold;
01222     #if DEBUG_OPTIMIZE
01223     fprintf (stderr, "optimize_merge_old: start\n");
01224     #endif
01225     anew = optimize->error_best;
01226     eold = optimize->error_old;
01227     i = j = k = 0;
01228     do
01229     {
01230         if (*enew < *eold)
01231         {
01232             memcpy (v + k * optimize->nvariables,
optimize->value
01233             + optimize->simulation_best[i] *

```

```

    optimize->nvariables,
01235         optimize->nvariables * sizeof (double));
01236     e[k] = *enew;
01237     ++k;
01238     ++enew;
01239     ++i;
01240 }
01241 else
01242 {
01243     memcpy (v + k * optimize->nvariables,
01244            optimize->value_old + j * optimize->
nvariables,
01245            optimize->nvariables * sizeof (double));
01246     e[k] = *eold;
01247     ++k;
01248     ++eold;
01249     ++j;
01250 }
01251 }
01252 while (k < optimize->nbest);
01253 memcpy (optimize->value_old, v, k * optimize->
nvariables * sizeof (double));
01254 memcpy (optimize->error_old, e, k * sizeof (double));
01255 #if DEBUG_OPTIMIZE
01256 fprintf (stderr, "optimize_merge_old: end\n");
01257 #endif
01258 }

```

4.21.2.15 optimize_MonteCarlo()

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

Function to optimize with the Monte-Carlo algorithm.

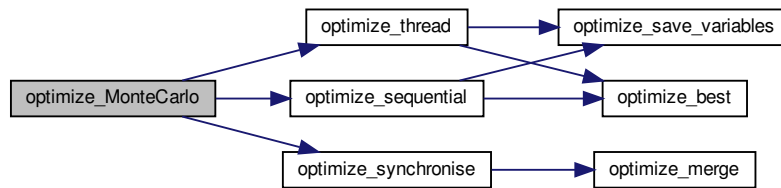
Definition at line 720 of file [optimize.c](#).

```

00721 {
00722     unsigned int i, j;
00723     GThread *thread[nthreads];
00724     ParallelData data[nthreads];
00725 #if DEBUG_OPTIMIZE
00726     fprintf (stderr, "optimize_MonteCarlo: start\n");
00727 #endif
00728     for (i = 0; i < optimize->nsimulations; ++i)
00729         for (j = 0; j < optimize->nvariables; ++j)
00730             optimize->value[i * optimize->nvariables + j]
00731                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->
rng)
00732                 * (optimize->rangemax[j] - optimize->rangemin[j]);
00733     optimize->nsaveds = 0;
00734     if (nthreads <= 1)
00735         optimize_sequential ();
00736     else
00737     {
00738         for (i = 0; i < nthreads; ++i)
00739         {
00740             data[i].thread = i;
00741             thread[i]
00742                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00743         }
00744         for (i = 0; i < nthreads; ++i)
00745             g_thread_join (thread[i]);
00746     }
00747 #if HAVE_MPI
00748     // Communicating tasks results
00749     optimize_synchronise ();
00750 #endif
00751 #if DEBUG_OPTIMIZE
00752     fprintf (stderr, "optimize_MonteCarlo: end\n");
00753 #endif
00754 }

```

Here is the call graph for this function:



4.21.2.16 optimize_norm_euclidian()

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

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

Parameters

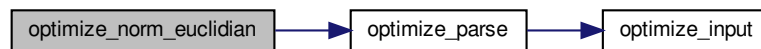
<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 297 of file [optimize.c](#).

```

00298 {
00299     double e, ei;
00300     unsigned int i;
00301     #if DEBUG_OPTIMIZE
00302     fprintf (stderr, "optimize_norm_euclidian: start\n");
00303     #endif
00304     e = 0.;
00305     for (i = 0; i < optimize->nexperiments; ++i)
00306     {
00307         ei = optimize_parse (simulation, i);
00308         e += ei * ei;
00309     }
00310     e = sqrt (e);
00311     #if DEBUG_OPTIMIZE
00312     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00313     fprintf (stderr, "optimize_norm_euclidian: end\n");
00314     #endif
00315     return e;
00316 }
```

Here is the call graph for this function:



4.21.2.17 optimize_norm_maximum()

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

Function to calculate the maximum error norm.

Returns

Maximum error norm.

Parameters

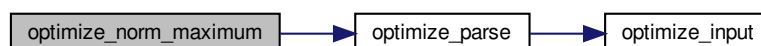
<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 324 of file [optimize.c](#).

```

00325 {
00326     double e, ei;
00327     unsigned int i;
00328     #if DEBUG_OPTIMIZE
00329     fprintf (stderr, "optimize_norm_maximum: start\n");
00330     #endif
00331     e = 0.;
00332     for (i = 0; i < optimize->nexperiments; ++i)
00333     {
00334         ei = fabs (optimize_parse (simulation, i));
00335         e = fmax (e, ei);
00336     }
00337     #if DEBUG_OPTIMIZE
00338     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00339     fprintf (stderr, "optimize_norm_maximum: end\n");
00340     #endif
00341     return e;
00342 }
```

Here is the call graph for this function:



4.21.2.18 optimize_norm_p()

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

Function to calculate the P error norm.

Returns

P error norm.

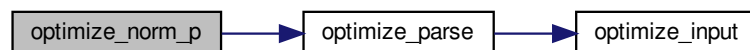
Parameters

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

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

```
00351 {
00352     double e, ei;
00353     unsigned int i;
00354     #if DEBUG_OPTIMIZE
00355     fprintf (stderr, "optimize_norm_p: start\n");
00356     #endif
00357     e = 0.;
00358     for (i = 0; i < optimize->nexperiments; ++i)
00359     {
00360         ei = fabs (optimize_parse (simulation, i));
00361         e += pow (ei, optimize->p);
00362     }
00363     e = pow (e, 1. / optimize->p);
00364     #if DEBUG_OPTIMIZE
00365     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00366     fprintf (stderr, "optimize_norm_p: end\n");
00367     #endif
00368     return e;
00369 }
```

Here is the call graph for this function:



4.21.2.19 optimize_norm_taxicab()

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

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

Parameters

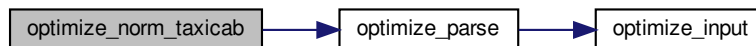
<i>simulation</i>	simulation number.
-------------------	--------------------

Definition at line 377 of file [optimize.c](#).

```

00378 {
00379     double e;
00380     unsigned int i;
00381     #if DEBUG_OPTIMIZE
00382     fprintf (stderr, "optimize_norm_taxicab: start\n");
00383     #endif
00384     e = 0.;
00385     for (i = 0; i < optimize->nexperiments; ++i)
00386         e += fabs (optimize_parse (simulation, i));
00387     #if DEBUG_OPTIMIZE
00388     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00389     fprintf (stderr, "optimize_norm_taxicab: end\n");
00390     #endif
00391     return e;
00392 }
```

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 1421 of file [optimize.c](#).

```

01422 {
01423     GTimeZone *tz;
01424     GDateTime *t0, *t;
01425     unsigned int i, j;
01426
01427     #if DEBUG_OPTIMIZE
01428     char *buffer;
01429     fprintf (stderr, "optimize_open: start\n");
01430     #endif
01431
01432     // Getting initial time
01433     #if DEBUG_OPTIMIZE
01434     fprintf (stderr, "optimize_open: getting initial time\n");
01435     #endif
01436     tz = g_time_zone_new_utc ();
01437     t0 = g_date_time_new_now (tz);
01438
01439     // Obtaining and initing the pseudo-random numbers generator seed
01440     #if DEBUG_OPTIMIZE
01441     fprintf (stderr, "optimize_open: getting initial seed\n");
01442     #endif
01443     if (optimize->seed == DEFAULT_RANDOM_SEED)
```

```

01444     optimize->seed = input->seed;
01445     gsl_rng_set (optimize->rng, optimize->seed);
01446
01447     // Replacing the working directory
01448     #if DEBUG_OPTIMIZE
01449     fprintf (stderr, "optimize_open: replacing the working directory\n");
01450     #endif
01451     g_chdir (input->directory);
01452
01453     // Getting results file names
01454     optimize->result = input->result;
01455     optimize->variables = input->variables;
01456
01457     // Obtaining the simulator file
01458     optimize->simulator = input->simulator;
01459
01460     // Obtaining the evaluator file
01461     optimize->evaluator = input->evaluator;
01462
01463     // Reading the algorithm
01464     optimize->algorithm = input->algorithm;
01465     switch (optimize->algorithm)
01466     {
01467     case ALGORITHM_MONTE_CARLO:
01468         optimize_algorithm = optimize_MonteCarlo;
01469         break;
01470     case ALGORITHM_SWEEP:
01471         optimize_algorithm = optimize_sweep;
01472         break;
01473     case ALGORITHM_ORTHOGONAL:
01474         optimize_algorithm = optimize_orthogonal;
01475         break;
01476     default:
01477         optimize_algorithm = optimize_genetic;
01478         optimize->mutation_ratio = input->
mutation_ratio;
01479         optimize->reproduction_ratio = input->
reproduction_ratio;
01480         optimize->adaptation_ratio = input->
adaptation_ratio;
01481     }
01482     optimize->nvariables = input->nvariables;
01483     optimize->nsimulations = input->nsimulations;
01484     optimize->niterations = input->niterations;
01485     optimize->nbest = input->nbest;
01486     optimize->tolerance = input->tolerance;
01487     optimize->nsteps = input->nsteps;
01488     optimize->nestimates = 0;
01489     optimize->threshold = input->threshold;
01490     optimize->stop = 0;
01491     if (input->nsteps)
01492     {
01493         optimize->relaxation = input->relaxation;
01494         switch (input->climbing)
01495         {
01496         case CLIMBING_METHOD_COORDINATES:
01497             optimize->nestimates = 2 * optimize->
nvariables;
01498             optimize_estimate_climbing =
optimize_estimate_climbing_coordinates;
01499             break;
01500         default:
01501             optimize->nestimates = input->nestimates;
01502             optimize_estimate_climbing =
optimize_estimate_climbing_random;
01503         }
01504     }
01505
01506     #if DEBUG_OPTIMIZE
01507     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01508     #endif
01509     optimize->simulation_best
01510     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01511     optimize->error_best = (double *) alloca (optimize->
nbest * sizeof (double));
01512
01513     // Reading the experimental data
01514     #if DEBUG_OPTIMIZE
01515     buffer = g_get_current_dir ();
01516     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01517     g_free (buffer);
01518     #endif
01519     optimize->nexperiments = input->nexperiments;
01520     optimize->ninputs = input->experiment->ninputs;
01521     optimize->experiment
01522     = (char **) alloca (input->nexperiments * sizeof (char *));
01523     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double)

```

```

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

```



```

01599     optimize_norm = optimize_norm_maximum;
01600     break;
01601     case ERROR_NORM_P:
01602         optimize_norm = optimize_norm_p;
01603         optimize->p = input->p;
01604         break;
01605     default:
01606         optimize_norm = optimize_norm_taxicab;
01607     }
01608
01609     // Allocating values
01610     #if DEBUG_OPTIMIZE
01611     fprintf (stderr, "optimize_open: allocating variables\n");
01612     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01613             optimize->nvariables, optimize->algorithm);
01614     #endif
01615     optimize->genetic_variable = NULL;
01616     if (optimize->algorithm == ALGORITHM_GENETIC)
01617     {
01618         optimize->genetic_variable = (GeneticVariable *)
01619             g_malloc (optimize->nvariables * sizeof (
01620             GeneticVariable));
01621         for (i = 0; i < optimize->nvariables; ++i)
01622         {
01623             #if DEBUG_OPTIMIZE
01624             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01625                     i, optimize->rangemin[i], optimize->
01626                     rangemax[i], optimize->nbits[i]);
01627             #endif
01628             optimize->genetic_variable[i].minimum =
01629                 optimize->rangemin[i];
01630             optimize->genetic_variable[i].maximum =
01631                 optimize->rangemax[i];
01632             optimize->genetic_variable[i].nbits = optimize->
01633                 nbits[i];
01634         }
01635     }
01636     #if DEBUG_OPTIMIZE
01637     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01638             optimize->nvariables, optimize->
01639             nsimulations);
01640     #endif
01641     optimize->value = (double *)
01642         g_malloc ((optimize->nsimulations
01643             + optimize->nestimates * optimize->
01644             nsteps)
01645             * optimize->nvariables * sizeof (double));
01646
01647     // Calculating simulations to perform for each task
01648     #if HAVE_MPI
01649     #if DEBUG_OPTIMIZE
01650     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01651             optimize->mpi_rank, ntasks);
01652     #endif
01653     optimize->nstart = optimize->mpi_rank * optimize->
01654         nsimulations / ntasks;
01655     optimize->nend = (1 + optimize->mpi_rank) *
01656         optimize->nsimulations / ntasks;
01657     if (optimize->nsteps)
01658     {
01659         optimize->nstart_climbing
01660             = optimize->mpi_rank * optimize->nestimates /
01661             ntasks;
01662         optimize->nend_climbing
01663             = (1 + optimize->mpi_rank) * optimize->
01664             nestimates / ntasks;
01665     }
01666     #else
01667     optimize->nstart = 0;
01668     optimize->nend = optimize->nsimulations;
01669     if (optimize->nsteps)
01670     {
01671         optimize->nstart_climbing = 0;
01672         optimize->nend_climbing = optimize->
01673             nestimates;
01674     }
01675     #endif
01676     #if DEBUG_OPTIMIZE
01677     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->
01678         nstart,
01679         optimize->nend);
01680     #endif
01681
01682     // Calculating simulations to perform for each thread
01683     optimize->thread
01684         = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));

```


4.21.2.21 optimize_orthogonal()

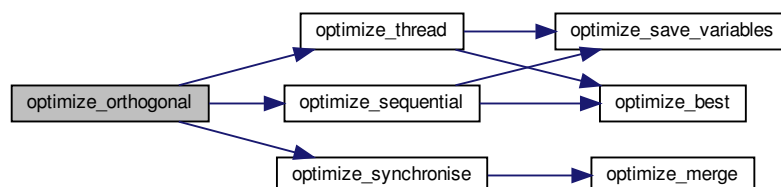
```
static void optimize_orthogonal ( ) [static]
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 760 of file [optimize.c](#).

```
00761 {
00762     unsigned int i, j, k, l;
00763     double e;
00764     GThread *thread[nthreads];
00765     ParallelData data[nthreads];
00766     #if DEBUG_OPTIMIZE
00767     fprintf (stderr, "optimize_orthogonal: start\n");
00768     #endif
00769     for (i = 0; i < optimize->nsimulations; ++i)
00770     {
00771         k = i;
00772         for (j = 0; j < optimize->nvariables; ++j)
00773         {
00774             l = k % optimize->nsweeps[j];
00775             k /= optimize->nsweeps[j];
00776             e = optimize->rangemin[j];
00777             if (optimize->nsweeps[j] > 1)
00778                 e += (l + gsl_rng_uniform (optimize->rng))
00779                     * (optimize->rangemax[j] - optimize->
00780                        rangemin[j])
00781                     / optimize->nsweeps[j];
00782             optimize->value[i * optimize->nvariables + j] = e;
00783         }
00784         optimize->nsaveds = 0;
00785         if (nthreads <= 1)
00786             optimize_sequential ();
00787         else
00788         {
00789             for (i = 0; i < nthreads; ++i)
00790             {
00791                 data[i].thread = i;
00792                 thread[i]
00793                     = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00794             }
00795             for (i = 0; i < nthreads; ++i)
00796                 g_thread_join (thread[i]);
00797         }
00798         #if HAVE_MPI
00799         // Communicating tasks results
00800         optimize_synchronise ();
00801         #endif
00802         #if DEBUG_OPTIMIZE
00803         fprintf (stderr, "optimize_orthogonal: end\n");
00804         #endif
00805     }
```

Here is the call graph for this function:



4.21.2.22 optimize_parse()

```
static double optimize_parse (
    unsigned int simulation,
    unsigned int experiment ) [static]
```

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

Returns

Objective function value.

Parameters

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

Definition at line 183 of file [optimize.c](#).

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

```

00237     snprintf (buffer, 512, "%s\" %s %s %s",
00238               buffer4, output, optimize->experiment[experiment], result);
00239     g_free (buffer4);
00240     g_free (buffer3);
00241     g_free (buffer2);
00242 #if DEBUG_OPTIMIZE
00243     fprintf (stderr, "optimize_parse: %s\n", buffer);
00244     fprintf (stderr, "optimize_parse: result=%s\n", result);
00245 #endif
00246     if (system (buffer) == -1)
00247         error_message = buffer;
00248     file_result = g_fopen (result, "r");
00249     e = atof (fgets (buffer, 512, file_result));
00250     fclose (file_result);
00251 }
00252 else
00253 {
00254 #if DEBUG_OPTIMIZE
00255     fprintf (stderr, "optimize_parse: output=%s\n", output);
00256 #endif
00257     strcpy (result, "");
00258     file_result = g_fopen (output, "r");
00259     e = atof (fgets (buffer, 512, file_result));
00260     fclose (file_result);
00261 }
00262 // Removing files
00263 #if !DEBUG_OPTIMIZE
00264 for (i = 0; i < optimize->ninputs; ++i)
00265 {
00266     if (optimize->file[i][0])
00267     {
00268         snprintf (buffer, 512, RM " %s", &input[i][0]);
00269         if (system (buffer) == -1)
00270             error_message = buffer;
00271     }
00272 }
00273 }
00274 snprintf (buffer, 512, RM " %s %s", output, result);
00275 if (system (buffer) == -1)
00276     error_message = buffer;
00277 #endif
00278 // Processing pending events
00279 if (show_pending)
00280     show_pending ();
00281 #if DEBUG_OPTIMIZE
00282 fprintf (stderr, "optimize_parse: end\n");
00283 #endif
00284 // Returning the objective function
00285 return e * optimize->weight[experiment];
00286 }

```

Here is the call graph for this function:



4.21.2.23 optimize_print()

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

Function to print the results.

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

```

00399 {
00400     unsigned int i;
00401     char buffer[512];
00402     #if HAVE_MPI
00403         if (optimize->mpi_rank)
00404             return;
00405     #endif
00406     printf ("%s\n", _("Best result"));
00407     fprintf (optimize->file_result, "%s\n", _("Best result"));
00408     printf ("error = %.15le\n", optimize->error_old[0]);
00409     fprintf (optimize->file_result, "error = %.15le\n",
optimize->error_old[0]);
00410     for (i = 0; i < optimize->nvariables; ++i)
00411     {
00412         snprintf (buffer, 512, "%s = %s\n",
optimize->label[i], format[optimize->
precision[i]]);
00414         printf (buffer, optimize->value_old[i]);
00415         fprintf (optimize->file_result, buffer, optimize->
value_old[i]);
00416     }
00417     fflush (optimize->file_result);
00418 }

```

4.21.2.24 optimize_refine()

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

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

Definition at line 1265 of file [optimize.c](#).

```

01266 {
01267     unsigned int i, j;
01268     double d;
01269     #if HAVE_MPI
01270         MPI_Status mpi_stat;
01271     #endif
01272     #if DEBUG_OPTIMIZE
01273         fprintf (stderr, "optimize_refine: start\n");
01274     #endif
01275     #if HAVE_MPI
01276         if (!optimize->mpi_rank)
01277         {
01278             #endif
01279             for (j = 0; j < optimize->nvariables; ++j)
01280             {
01281                 optimize->rangemin[j] = optimize->rangemax[j]
= optimize->value_old[j];
01282             }
01283             for (i = 0; ++i < optimize->nbest; )
01284             {
01285                 for (j = 0; j < optimize->nvariables; ++j)
01286                 {
01287                     optimize->rangemin[j]
= fmin (optimize->rangemin[j],
optimize->value_old[i * optimize->
nvariables + j]);
01291                     optimize->rangemax[j]
= fmax (optimize->rangemax[j],
optimize->value_old[i * optimize->
nvariables + j]);
01294                 }
01295             }
01296             for (j = 0; j < optimize->nvariables; ++j)
01297             {
01298                 d = optimize->tolerance
* (optimize->rangemax[j] - optimize->
rangemin[j]);
01300                 switch (optimize->algorithm)
01301                 {
01302                     case ALGORITHM_MONTE_CARLO:
01303                         d *= 0.5;
01304                         break;
01305                     default:

```

```

01306         if (optimize->nsweeps[j] > 1)
01307             d /= optimize->nsweeps[j] - 1;
01308         else
01309             d = 0.;
01310     }
01311     optimize->rangemin[j] -= d;
01312     optimize->rangemin[j]
01313     = fmax (optimize->rangemin[j], optimize->
rangeminabs[j]);
01314     optimize->rangemax[j] += d;
01315     optimize->rangemax[j]
01316     = fmin (optimize->rangemax[j], optimize->
rangemaxabs[j]);
01317     printf ("%s min=%lg max=%lg\n", optimize->label[j],
optimize->rangemin[j], optimize->
rangemax[j]);
01319     fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
optimize->label[j], optimize->rangemin[j],
optimize->rangemax[j]);
01322     }
01323     #if HAVE_MPI
01324     for (i = 1; (int) i < ntasks; ++i)
01325     {
01326         MPI_Send (optimize->rangemin, optimize->
nvariables, MPI_DOUBLE, i,
01327                 1, MPI_COMM_WORLD);
01328         MPI_Send (optimize->rangemax, optimize->
nvariables, MPI_DOUBLE, i,
01329                 1, MPI_COMM_WORLD);
01330     }
01331     }
01332     else
01333     {
01334         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
1,
01335                 MPI_COMM_WORLD, &mpi_stat);
01336         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
1,
01337                 MPI_COMM_WORLD, &mpi_stat);
01338     }
01339     #endif
01340     #if DEBUG_OPTIMIZE
01341     fprintf (stderr, "optimize_refine: end\n");
01342     #endif
01343 }

```

4.21.2.25 optimize_save_old()

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

Function to save the best results on iterative methods.

Definition at line 1185 of file optimize.c.

```

01186 {
01187     unsigned int i, j;
01188     #if DEBUG_OPTIMIZE
01189     fprintf (stderr, "optimize_save_old: start\n");
01190     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01191     #endif
01192     memcpy (optimize->error_old, optimize->error_best,
optimize->nbest * sizeof (double));
01193     for (i = 0; i < optimize->nbest; ++i)
01194     {
01195         j = optimize->simulation_best[i];
01196         #if DEBUG_OPTIMIZE
01197         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01198         #endif
01199     }
01200     memcpy (optimize->value_old + i * optimize->
nvariables,
01201            optimize->value + j * optimize->nvariables,
optimize->nvariables * sizeof (double));
01202     }
01203     #if DEBUG_OPTIMIZE
01204     for (i = 0; i < optimize->nvariables; ++i)
01205     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
i, optimize->value_old[i]);
01206     fprintf (stderr, "optimize_save_old: end\n");
01207     #endif
01208 }

```

4.21.2.26 optimize_save_variables()

```
static void optimize_save_variables (
    unsigned int simulation,
    double error ) [static]
```

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

Parameters

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

Definition at line 424 of file [optimize.c](#).

```
00426 {
00427     unsigned int i;
00428     char buffer[64];
00429     #if DEBUG_OPTIMIZE
00430     fprintf (stderr, "optimize_save_variables: start\n");
00431     #endif
00432     for (i = 0; i < optimize->nvariables; ++i)
00433     {
00434         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00435         fprintf (optimize->file_variables, buffer,
00436                 optimize->value[simulation * optimize->
nvariables + i]);
00437     }
00438     fprintf (optimize->file_variables, "%.14le\n", error);
00439     fflush (optimize->file_variables);
00440     #if DEBUG_OPTIMIZE
00441     fprintf (stderr, "optimize_save_variables: end\n");
00442     #endif
00443 }
```

4.21.2.27 optimize_sequential()

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

Function to optimize sequentially.

Definition at line 490 of file [optimize.c](#).

```
00491 {
00492     unsigned int i;
00493     double e;
00494     #if DEBUG_OPTIMIZE
00495     fprintf (stderr, "optimize_sequential: start\n");
00496     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00497             optimize->nstart, optimize->nend);
00498     #endif
00499     for (i = optimize->nstart; i < optimize->nend; ++i)
00500     {
00501         e = optimize_norm (i);
00502         optimize_best (i, e);
00503         optimize_save_variables (i, e);
00504         if (e < optimize->threshold)
00505         {
00506             optimize->stop = 1;
```

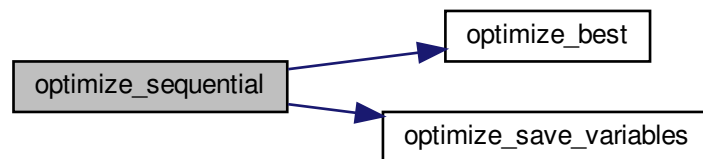


```

00507         break;
00508     }
00509     #if DEBUG_OPTIMIZE
00510     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00511     #endif
00512 }
00513 #if DEBUG_OPTIMIZE
00514     fprintf (stderr, "optimize_sequential: end\n");
00515 #endif
00516 }

```

Here is the call graph for this function:



4.21.2.28 optimize_step()

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

Function to do a step of the iterative algorithm.

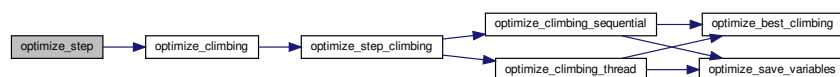
Definition at line 1349 of file `optimize.c`.

```

01350 {
01351     #if DEBUG_OPTIMIZE
01352     fprintf (stderr, "optimize_step: start\n");
01353     #endif
01354     optimize_algorithm ();
01355     if (optimize->nsteps)
01356         optimize_climbing ();
01357     #if DEBUG_OPTIMIZE
01358     fprintf (stderr, "optimize_step: end\n");
01359     #endif
01360 }

```

Here is the call graph for this function:



4.21.2.29 optimize_step_climbing()

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

Function to do a step of the hill climbing method.

Parameters

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

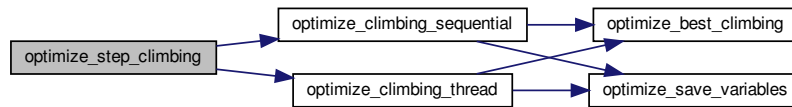
Definition at line 970 of file `optimize.c`.

```

00971 {
00972     GThread *thread[nthreads_climbing];
00973     ParallelData data[nthreads_climbing];
00974     unsigned int i, j, k, b;
00975     #if DEBUG_OPTIMIZE
00976     fprintf (stderr, "optimize_step_climbing: start\n");
00977     #endif
00978     for (i = 0; i < optimize->nestimates; ++i)
00979     {
00980         k = (simulation + i) * optimize->nvariables;
00981         b = optimize->simulation_best[0] * optimize->
nvariables;
00982     #if DEBUG_OPTIMIZE
00983         fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
00984                 simulation + i, optimize->simulation_best[0]);
00985     #endif
00986         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00987         {
00988             #if DEBUG_OPTIMIZE
00989             fprintf (stderr,
00990                     "optimize_step_climbing: estimate=%u best%u=%.14le\n",
00991                     i, j, optimize->value[b]);
00992             #endif
00993             optimize->value[k]
00994             = optimize->value[b] + optimize_estimate_climbing (j, i)
;
00995             optimize->value[k] = fmin (fmax (optimize->value[k],
00996                                             optimize->rangeminabs[j]),
00997                                       optimize->rangemaxabs[j]);
00998             #if DEBUG_OPTIMIZE
00999             fprintf (stderr,
01000                     "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
01001                     i, j, optimize->value[k]);
01002             #endif
01003         }
01004     }
01005     if (nthreads_climbing == 1)
01006         optimize_climbing_sequential (simulation);
01007     else
01008     {
01009         for (i = 0; i <= nthreads_climbing; ++i)
01010         {
01011             optimize->thread_climbing[i]
01012             = simulation + optimize->nstart_climbing
01013             + i * (optimize->nend_climbing - optimize->
nstart_climbing)
01014             / nthreads_climbing;
01015             #if DEBUG_OPTIMIZE
01016             fprintf (stderr,
01017                     "optimize_step_climbing: i=%u thread_climbing=%u\n",
01018                     i, optimize->thread_climbing[i]);
01019             #endif
01020         }
01021         for (i = 0; i < nthreads_climbing; ++i)
01022         {
01023             data[i].thread = i;
01024             thread[i] = g_thread_new
01025             (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01026         }
01027         for (i = 0; i < nthreads_climbing; ++i)
01028             g_thread_join (thread[i]);
01029     }
01030     #if DEBUG_OPTIMIZE
01031     fprintf (stderr, "optimize_step_climbing: end\n");
01032     #endif
01033 }

```

Here is the call graph for this function:



4.21.2.30 optimize_sweep()

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

Function to optimize with the sweep algorithm.

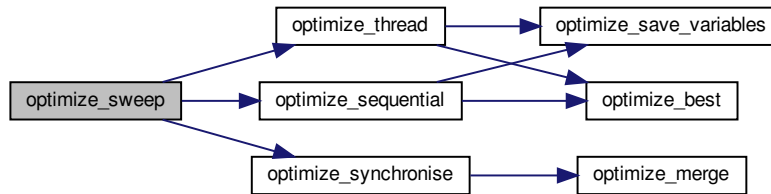
Definition at line 670 of file `optimize.c`.

```

00671 {
00672     unsigned int i, j, k, l;
00673     double e;
00674     GThread *thread[nthreads];
00675     ParallelData data[nthreads];
00676     #if DEBUG_OPTIMIZE
00677         fprintf (stderr, "optimize_sweep: start\n");
00678     #endif
00679     for (i = 0; i < optimize->nsimulations; ++i)
00680     {
00681         k = i;
00682         for (j = 0; j < optimize->nvariables; ++j)
00683         {
00684             l = k % optimize->nsweeps[j];
00685             k /= optimize->nsweeps[j];
00686             e = optimize->rangemin[j];
00687             if (optimize->nsweeps[j] > 1)
00688                 e += l * (optimize->rangemax[j] - optimize->
rangemin[j])
00689                     / (optimize->nsweeps[j] - 1);
00690             optimize->value[i * optimize->nvariables + j] = e;
00691         }
00692     }
00693     optimize->nsaveds = 0;
00694     if (nthreads <= 1)
00695         optimize_sequential ();
00696     else
00697     {
00698         for (i = 0; i < nthreads; ++i)
00699         {
00700             data[i].thread = i;
00701             thread[i]
= g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00703         }
00704         for (i = 0; i < nthreads; ++i)
00705             g_thread_join (thread[i]);
00706     }
00707     #if HAVE_MPI
00708         // Communicating tasks results
00709         optimize_synchronise ();
00710     #endif
00711     #if DEBUG_OPTIMIZE
00712         fprintf (stderr, "optimize_sweep: end\n");
00713     #endif
00714 }

```

Here is the call graph for this function:



4.21.2.31 optimize_synchronise()

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

Function to synchronise the optimization results of MPI tasks.

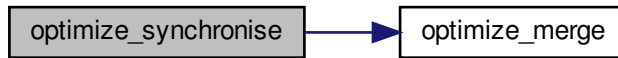
Definition at line 623 of file [optimize.c](#).

```

00624 {
00625     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00626     double error_best[optimize->nbest];
00627     MPI_Status mpi_stat;
00628     #if DEBUG_OPTIMIZE
00629     fprintf (stderr, "optimize_synchronise: start\n");
00630     #endif
00631     if (optimize->mpi_rank == 0)
00632     {
00633         for (i = 1; (int) i < ntasks; ++i)
00634         {
00635             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00636             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00637                     MPI_COMM_WORLD, &mpi_stat);
00638             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00639                     MPI_COMM_WORLD, &mpi_stat);
00640             optimize_merge (nsaveds, simulation_best, error_best);
00641             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00642             if (stop)
00643                 optimize->stop = 1;
00644         }
00645         for (i = 1; (int) i < ntasks; ++i)
00646             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00647     }
00648     else
00649     {
00650         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00651         MPI_Send (optimize->simulation_best, optimize->
00652                 nsaveds, MPI_INT, 0, 1,
00653                 MPI_COMM_WORLD);
00654         MPI_Send (optimize->error_best, optimize->
00655                 nsaveds, MPI_DOUBLE, 0, 1,
00656                 MPI_COMM_WORLD);
00657         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00658         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00659         if (stop)
00660             optimize->stop = 1;
00661     }
00662     #if DEBUG_OPTIMIZE
00663     fprintf (stderr, "optimize_synchronise: end\n");
00664     #endif
00665 }

```

Here is the call graph for this function:



4.21.2.32 optimize_thread()

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

Function to optimize on a thread.

Returns

NULL.

Parameters

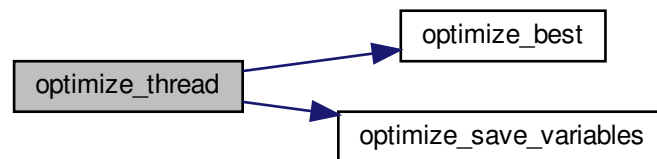
<i>data</i>	Function data.
-------------	----------------

Definition at line 524 of file [optimize.c](#).

```

00525 {
00526     unsigned int i, thread;
00527     double e;
00528     #if DEBUG_OPTIMIZE
00529     fprintf (stderr, "optimize_thread: start\n");
00530     #endif
00531     thread = data->thread;
00532     #if DEBUG_OPTIMIZE
00533     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00534             optimize->thread[thread], optimize->thread[thread + 1]);
00535     #endif
00536     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00537     {
00538         e = optimize_norm (i);
00539         g_mutex_lock (mutex);
00540         optimize_best (i, e);
00541         optimize_save_variables (i, e);
00542         if (e < optimize->threshold)
00543             optimize->stop = 1;
00544         g_mutex_unlock (mutex);
00545         if (optimize->stop)
00546             break;
00547     #if DEBUG_OPTIMIZE
00548     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00549     #endif
00550     }
00551     #if DEBUG_OPTIMIZE
00552     fprintf (stderr, "optimize_thread: end\n");
00553     #endif
00554     g_thread_exit (NULL);
00555     return NULL;
00556 }
```

Here is the call graph for this function:



4.22 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 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 #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"

```

```

00064 #include "optimize.h"
00065
00066 #define DEBUG_OPTIMIZE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define RM "del"
00074 #else
00075 #define RM "rm"
00076 #endif
00077
00078 Optimize optimize[1];
00079 unsigned int nthreads_climbing;
00081
00082 static void (*optimize_algorithm) ();
00084 static double (*optimize_estimate_climbing) (unsigned int variable,
00085                                             unsigned int estimate);
00087 static double (*optimize_norm) (unsigned int simulation);
00089
00093 static inline void
00094 optimize_input (unsigned int simulation,
00095               char *input,
00096               GMappedFile * stencil)
00097 {
00098     char buffer[32], value[32];
00099     GRegex *regex;
00100     FILE *file;
00101     char *buffer2, *buffer3 = NULL, *content;
00102     gsize length;
00103     unsigned int i;
00104
00105     #if DEBUG_OPTIMIZE
00106     fprintf (stderr, "optimize_input: start\n");
00107     #endif
00108
00109     // Checking the file
00110     if (!stencil)
00111         goto optimize_input_end;
00112
00113     // Opening stencil
00114     content = g_mapped_file_get_contents (stencil);
00115     length = g_mapped_file_get_length (stencil);
00116     #if DEBUG_OPTIMIZE
00117     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00118     #endif
00119     file = g_fopen (input, "w");
00120
00121     // Parsing stencil
00122     for (i = 0; i < optimize->nvariables; ++i)
00123     {
00124         #if DEBUG_OPTIMIZE
00125         fprintf (stderr, "optimize_input: variable=%u\n", i);
00126         #endif
00127         snprintf (buffer, 32, "@variable%u@", i + 1);
00128         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00129                             NULL);
00130         if (i == 0)
00131         {
00132             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00133                                              optimize->label[i],
00134                                              (GRegexMatchFlags) 0, NULL);
00135             #if DEBUG_OPTIMIZE
00136             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00137             #endif
00138         }
00139         else
00140         {
00141             length = strlen (buffer3);
00142             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00143                                              optimize->label[i],
00144                                              (GRegexMatchFlags) 0, NULL);
00145             g_free (buffer3);
00146         }
00147         g_regex_unref (regex);
00148         length = strlen (buffer2);
00149         snprintf (buffer, 32, "@value%u@", i + 1);
00150         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00151                             NULL);
00152         snprintf (value, 32, format[optimize->precision[i]],
00153                 optimize->value[simulation * optimize->nvariables + i]);
00154
00155         #if DEBUG_OPTIMIZE
00156         fprintf (stderr, "optimize_input: value=%s\n", value);
00157         #endif
00158         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00159                                          (GRegexMatchFlags) 0, NULL);
00160         g_free (buffer2);

```

```

00161     g_regex_unref (regex);
00162 }
00163
00164 // Saving input file
00165 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00166 g_free (buffer3);
00167 fclose (file);
00168
00169 optimize_input_end:
00170 #if DEBUG_OPTIMIZE
00171     fprintf (stderr, "optimize_input: end\n");
00172 #endif
00173     return;
00174 }
00175
00182 static double
00183 optimize_parse (unsigned int simulation,
00184                 unsigned int experiment)
00185 {
00186     unsigned int i;
00187     double e;
00188     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00189           *buffer3, *buffer4;
00190     FILE *file_result;
00191
00192     #if DEBUG_OPTIMIZE
00193         fprintf (stderr, "optimize_parse: start\n");
00194         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00195                 simulation, experiment);
00196     #endif
00197
00198     // Opening input files
00199     for (i = 0; i < optimize->ninputs; ++i)
00200     {
00201         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00202         #if DEBUG_OPTIMIZE
00203             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00204         #endif
00205         optimize_input (simulation, &input[i][0], optimize->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     if (system (buffer) == -1)
00228         error_message = buffer;
00229
00230     // Checking the objective value function
00231     if (optimize->evaluator)
00232     {
00233         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00234         buffer2 = g_path_get_dirname (optimize->evaluator);
00235         buffer3 = g_path_get_basename (optimize->evaluator);
00236         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00237         snprintf (buffer, 512, "%s\\%s %s %s %s",
00238                 buffer4, output, optimize->experiment[experiment], result);
00239         g_free (buffer4);
00240         g_free (buffer3);
00241         g_free (buffer2);
00242         #if DEBUG_OPTIMIZE
00243             fprintf (stderr, "optimize_parse: %s\n", buffer);
00244             fprintf (stderr, "optimize_parse: result=%s\n", result);
00245         #endif
00246         if (system (buffer) == -1)
00247             error_message = buffer;
00248         file_result = g_fopen (result, "r");
00249         e = atof (fgets (buffer, 512, file_result));
00250         fclose (file_result);
00251     }
00252     else
00253     {

```



```

00254 #if DEBUG_OPTIMIZE
00255     fprintf (stderr, "optimize_parse: output=%s\n", output);
00256 #endif
00257     strcpy (result, "");
00258     file_result = g_fopen (output, "r");
00259     e = atof (fgets (buffer, 512, file_result));
00260     fclose (file_result);
00261 }
00262
00263 // Removing files
00264 #if !DEBUG_OPTIMIZE
00265     for (i = 0; i < optimize->ninputs; ++i)
00266     {
00267         if (optimize->file[i][0])
00268         {
00269             snprintf (buffer, 512, RM " %s", &input[i][0]);
00270             if (system (buffer) == -1)
00271                 error_message = buffer;
00272         }
00273     }
00274     snprintf (buffer, 512, RM " %s %s", output, result);
00275     if (system (buffer) == -1)
00276         error_message = buffer;
00277 #endif
00278
00279 // Processing pending events
00280 if (show_pending)
00281     show_pending ();
00282
00283 #if DEBUG_OPTIMIZE
00284     fprintf (stderr, "optimize_parse: end\n");
00285 #endif
00286
00287 // Returning the objective function
00288     return e * optimize->weight[experiment];
00289 }
00290
00291 static 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
00313 static double
00314 optimize_norm_maximum (unsigned int simulation)
00315 {
00316     double e, ei;
00317     unsigned int i;
00318     #if DEBUG_OPTIMIZE
00319         fprintf (stderr, "optimize_norm_maximum: start\n");
00320     #endif
00321     e = 0.;
00322     for (i = 0; i < optimize->nexperiments; ++i)
00323     {
00324         ei = fabs (optimize_parse (simulation, i));
00325         e = fmax (e, ei);
00326     }
00327     #if DEBUG_OPTIMIZE
00328         fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00329         fprintf (stderr, "optimize_norm_maximum: end\n");
00330     #endif
00331     return e;
00332 }
00333
00334 static double
00335 optimize_norm_p (unsigned int simulation)
00336 {
00337     double e, ei;
00338     unsigned int i;
00339     #if DEBUG_OPTIMIZE
00340         fprintf (stderr, "optimize_norm_p: start\n");

```

```

00356 #endif
00357     e = 0.;
00358     for (i = 0; i < optimize->nexperiments; ++i)
00359     {
00360         ei = fabs (optimize_parse (simulation, i));
00361         e += pow (ei, optimize->p);
00362     }
00363     e = pow (e, 1. / optimize->p);
00364     #if DEBUG_OPTIMIZE
00365     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00366     fprintf (stderr, "optimize_norm_p: end\n");
00367 #endif
00368     return e;
00369 }
00370
00376 static double
00377 optimize_norm_taxicab (unsigned int simulation)
00378 {
00379     double e;
00380     unsigned int i;
00381     #if DEBUG_OPTIMIZE
00382     fprintf (stderr, "optimize_norm_taxicab: start\n");
00383 #endif
00384     e = 0.;
00385     for (i = 0; i < optimize->nexperiments; ++i)
00386         e += fabs (optimize_parse (simulation, i));
00387     #if DEBUG_OPTIMIZE
00388     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00389     fprintf (stderr, "optimize_norm_taxicab: end\n");
00390 #endif
00391     return e;
00392 }
00393
00397 static void
00398 optimize_print ()
00399 {
00400     unsigned int i;
00401     char buffer[512];
00402     #if HAVE_MPI
00403     if (optimize->mpi_rank)
00404         return;
00405 #endif
00406     printf ("%s\n", _("Best result"));
00407     fprintf (optimize->file_result, "%s\n", _("Best result"));
00408     printf ("error = %.15le\n", optimize->error_old[0]);
00409     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00410     for (i = 0; i < optimize->nvariables; ++i)
00411     {
00412         snprintf (buffer, 512, "%s = %s\n",
00413                 optimize->label[i], format[optimize->precision[i]]);
00414         printf (buffer, optimize->value_old[i]);
00415         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00416     }
00417     fflush (optimize->file_result);
00418 }
00419
00423 static void
00424 optimize_save_variables (unsigned int simulation,
00425                         double error)
00426 {
00427     unsigned int i;
00428     char buffer[64];
00429     #if DEBUG_OPTIMIZE
00430     fprintf (stderr, "optimize_save_variables: start\n");
00431 #endif
00432     for (i = 0; i < optimize->nvariables; ++i)
00433     {
00434         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00435         fprintf (optimize->file_variables, buffer,
00436                 optimize->value[simulation * optimize->nvariables + i]);
00437     }
00438     fprintf (optimize->file_variables, "%.14le\n", error);
00439     fflush (optimize->file_variables);
00440     #if DEBUG_OPTIMIZE
00441     fprintf (stderr, "optimize_save_variables: end\n");
00442 #endif
00443 }
00444
00448 static void
00449 optimize_best (unsigned int simulation,
00450               double value)
00451 {
00452     unsigned int i, j;
00453     double e;
00454     #if DEBUG_OPTIMIZE
00455     fprintf (stderr, "optimize_best: start\n");

```

```

00456     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00457               optimize->nsaveds, optimize->nbest);
00458 #endif
00459     if (optimize->nsaveds < optimize->nbest
00460         || value < optimize->error_best[optimize->nsaveds - 1])
00461     {
00462         if (optimize->nsaveds < optimize->nbest)
00463             ++optimize->nsaveds;
00464         optimize->error_best[optimize->nsaveds - 1] = value;
00465         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00466         for (i = optimize->nsaveds; --i;)
00467         {
00468             if (optimize->error_best[i] < optimize->error_best[i - 1])
00469             {
00470                 j = optimize->simulation_best[i];
00471                 e = optimize->error_best[i];
00472                 optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00473                 optimize->error_best[i] = optimize->error_best[i - 1];
00474                 optimize->simulation_best[i - 1] = j;
00475                 optimize->error_best[i - 1] = e;
00476             }
00477             else
00478                 break;
00479         }
00480     }
00481 #if DEBUG_OPTIMIZE
00482     fprintf (stderr, "optimize_best: end\n");
00483 #endif
00484 }
00485
00486 static void
00490 optimize_sequential ()
00491 {
00492     unsigned int i;
00493     double e;
00494 #if DEBUG_OPTIMIZE
00495     fprintf (stderr, "optimize_sequential: start\n");
00496     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00497             optimize->nstart, optimize->nend);
00498 #endif
00499     for (i = optimize->nstart; i < optimize->nend; ++i)
00500     {
00501         e = optimize_norm (i);
00502         optimize_best (i, e);
00503         optimize_save_variables (i, e);
00504         if (e < optimize->threshold)
00505         {
00506             optimize->stop = 1;
00507             break;
00508         }
00509 #if DEBUG_OPTIMIZE
00510         fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00511 #endif
00512     }
00513 #if DEBUG_OPTIMIZE
00514     fprintf (stderr, "optimize_sequential: end\n");
00515 #endif
00516 }
00517
00523 static void *
00524 optimize_thread (ParallelData * data)
00525 {
00526     unsigned int i, thread;
00527     double e;
00528 #if DEBUG_OPTIMIZE
00529     fprintf (stderr, "optimize_thread: start\n");
00530 #endif
00531     thread = data->thread;
00532 #if DEBUG_OPTIMIZE
00533     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00534             optimize->thread[thread], optimize->thread[thread + 1]);
00535 #endif
00536     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00537     {
00538         e = optimize_norm (i);
00539         g_mutex_lock (mutex);
00540         optimize_best (i, e);
00541         optimize_save_variables (i, e);
00542         if (e < optimize->threshold)
00543         {
00544             optimize->stop = 1;
00545             g_mutex_unlock (mutex);
00546             break;
00547         }
00548 #if DEBUG_OPTIMIZE
00549         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00550 #endif
00551     }
00552 }

```

```

00550     }
00551     #if DEBUG_OPTIMIZE
00552     fprintf (stderr, "optimize_thread: end\n");
00553     #endif
00554     g_thread_exit (NULL);
00555     return NULL;
00556 }
00557
00561 static inline void
00562 optimize_merge (unsigned int nsaveds,
00563                unsigned int *simulation_best,
00564                double *error_best)
00565 {
00566     unsigned int i, j, k, s[optimize->nbest];
00567     double e[optimize->nbest];
00568     #if DEBUG_OPTIMIZE
00569     fprintf (stderr, "optimize_merge: start\n");
00570     #endif
00571     i = j = k = 0;
00572     do
00573     {
00574         if (i == optimize->nsaveds)
00575         {
00576             s[k] = simulation_best[j];
00577             e[k] = error_best[j];
00578             ++j;
00579             ++k;
00580             if (j == nsaveds)
00581                 break;
00582         }
00583         else if (j == nsaveds)
00584         {
00585             s[k] = optimize->simulation_best[i];
00586             e[k] = optimize->error_best[i];
00587             ++i;
00588             ++k;
00589             if (i == optimize->nsaveds)
00590                 break;
00591         }
00592         else if (optimize->error_best[i] > error_best[j])
00593         {
00594             s[k] = simulation_best[j];
00595             e[k] = error_best[j];
00596             ++j;
00597             ++k;
00598         }
00599         else
00600         {
00601             s[k] = optimize->simulation_best[i];
00602             e[k] = optimize->error_best[i];
00603             ++i;
00604             ++k;
00605         }
00606     }
00607     while (k < optimize->nbest);
00608     optimize->nsaveds = k;
00609     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00610     memcpy (optimize->error_best, e, k * sizeof (double));
00611     #if DEBUG_OPTIMIZE
00612     fprintf (stderr, "optimize_merge: end\n");
00613     #endif
00614 }
00615
00621 #if HAVE_MPI
00622 static void
00623 optimize_synchronise ()
00624 {
00625     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00626     double error_best[optimize->nbest];
00627     MPI_Status mpi_stat;
00628     #if DEBUG_OPTIMIZE
00629     fprintf (stderr, "optimize_synchronise: start\n");
00630     #endif
00631     if (optimize->mpi_rank == 0)
00632     {
00633         for (i = 1; (int) i < ntasks; ++i)
00634         {
00635             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00636             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00637                      MPI_COMM_WORLD, &mpi_stat);
00638             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00639                      MPI_COMM_WORLD, &mpi_stat);
00640             optimize_merge (nsaveds, simulation_best, error_best);
00641             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00642             if (stop)
00643                 optimize->stop = 1;
00644         }

```

```

00645     for (i = 1; (int) i < ntasks; ++i)
00646         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00647     }
00648     else
00649     {
00650         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00651         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00652             MPI_COMM_WORLD);
00653         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00654             MPI_COMM_WORLD);
00655         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00656         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00657         if (stop)
00658             optimize->stop = 1;
00659     }
00660 #if DEBUG_OPTIMIZE
00661     fprintf (stderr, "optimize_synchronise: end\n");
00662 #endif
00663 }
00664 #endif
00665
00666 static void
00667 optimize_sweep ()
00668 {
00669     unsigned int i, j, k, l;
00670     double e;
00671     GThread *thread[nthreads];
00672     ParallelData data[nthreads];
00673 #if DEBUG_OPTIMIZE
00674     fprintf (stderr, "optimize_sweep: start\n");
00675 #endif
00676     for (i = 0; i < optimize->nsimulations; ++i)
00677     {
00678         k = i;
00679         for (j = 0; j < optimize->nvariables; ++j)
00680         {
00681             l = k % optimize->nsweeps[j];
00682             k /= optimize->nsweeps[j];
00683             e = optimize->rangemin[j];
00684             if (optimize->nsweeps[j] > 1)
00685                 e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00686                     / (optimize->nsweeps[j] - 1);
00687             optimize->value[i * optimize->nvariables + j] = e;
00688         }
00689     }
00690     optimize->nsaveds = 0;
00691     if (nthreads <= 1)
00692         optimize_sequential ();
00693     else
00694     {
00695         for (i = 0; i < nthreads; ++i)
00696         {
00697             data[i].thread = i;
00698             thread[i]
00699                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00700         }
00701         for (i = 0; i < nthreads; ++i)
00702             g_thread_join (thread[i]);
00703     }
00704 #if HAVE_MPI
00705     // Communicating tasks results
00706     optimize_synchronise ();
00707 #endif
00708 #if DEBUG_OPTIMIZE
00709     fprintf (stderr, "optimize_sweep: end\n");
00710 #endif
00711 }
00712
00713 static void
00714 optimize_MonteCarlo ()
00715 {
00716     unsigned int i, j;
00717     GThread *thread[nthreads];
00718     ParallelData data[nthreads];
00719 #if DEBUG_OPTIMIZE
00720     fprintf (stderr, "optimize_MonteCarlo: start\n");
00721 #endif
00722     for (i = 0; i < optimize->nsimulations; ++i)
00723     {
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         {

```

```

00738     for (i = 0; i < nthreads; ++i)
00739     {
00740         data[i].thread = i;
00741         thread[i]
00742             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00743     }
00744     for (i = 0; i < nthreads; ++i)
00745         g_thread_join (thread[i]);
00746 }
00747 #if HAVE_MPI
00748 // Communicating tasks results
00749 optimize_synchronise ();
00750 #endif
00751 #if DEBUG_OPTIMIZE
00752 fprintf (stderr, "optimize_MonteCarlo: end\n");
00753 #endif
00754 }
00755
00756 static void
00757 optimize_orthogonal ()
00758 {
00759     unsigned int i, j, k, l;
00760     double e;
00761     GThread *thread[nthreads];
00762     ParallelData data[nthreads];
00763 #if DEBUG_OPTIMIZE
00764 fprintf (stderr, "optimize_orthogonal: start\n");
00765 #endif
00766 for (i = 0; i < optimize->nsimulations; ++i)
00767 {
00768     k = i;
00769     for (j = 0; j < optimize->nvariables; ++j)
00770     {
00771         l = k % optimize->nsweeps[j];
00772         k /= optimize->nsweeps[j];
00773         e = optimize->rangemin[j];
00774         if (optimize->nsweeps[j] > 1)
00775             e += (l + gsl_rng_uniform (optimize->rng))
00776                 * (optimize->rangemax[j] - optimize->rangemin[j])
00777                 / optimize->nsweeps[j];
00778         optimize->value[i * optimize->nvariables + j] = e;
00779     }
00780 }
00781 optimize->nsaveds = 0;
00782 if (nthreads <= 1)
00783     optimize_sequential ();
00784 else
00785 {
00786     for (i = 0; i < nthreads; ++i)
00787     {
00788         data[i].thread = i;
00789         thread[i]
00790             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00791     }
00792     for (i = 0; i < nthreads; ++i)
00793         g_thread_join (thread[i]);
00794 }
00795 #if HAVE_MPI
00796 // Communicating tasks results
00797 optimize_synchronise ();
00798 #endif
00799 #if DEBUG_OPTIMIZE
00800 fprintf (stderr, "optimize_orthogonal: end\n");
00801 #endif
00802 }
00803
00804 static void
00805 optimize_best_climbing (unsigned int simulation,
00806                        double value)
00807 {
00808 #if DEBUG_OPTIMIZE
00809 fprintf (stderr, "optimize_best_climbing: start\n");
00810 fprintf (stderr,
00811         "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
00812         simulation, value, optimize->error_best[0]);
00813 #endif
00814 if (value < optimize->error_best[0])
00815 {
00816     optimize->error_best[0] = value;
00817     optimize->simulation_best[0] = simulation;
00818 #if DEBUG_OPTIMIZE
00819 fprintf (stderr,
00820         "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
00821         simulation, value);
00822 #endif
00823 }
00824 #if DEBUG_OPTIMIZE
00825 }
00826 #endif

```

```

00831     fprintf (stderr, "optimize_best_climbing: end\n");
00832 #endif
00833 }
00834
00835 static inline void
00836 optimize_climbing_sequential (unsigned int simulation)
00837 {
00838     double e;
00839     unsigned int i, j;
00840 #if DEBUG_OPTIMIZE
00841     fprintf (stderr, "optimize_climbing_sequential: start\n");
00842     fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
00843             "nend_climbing=%u\n",
00844             optimize->nstart_climbing, optimize->nend_climbing);
00845 #endif
00846     for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
00847     {
00848         j = simulation + i;
00849         e = optimize_norm (j);
00850         optimize_best_climbing (j, e);
00851         optimize_save_variables (j, e);
00852         if (e < optimize->threshold)
00853         {
00854             optimize->stop = 1;
00855             break;
00856         }
00857 #if DEBUG_OPTIMIZE
00858     fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
00859 #endif
00860     }
00861 #if DEBUG_OPTIMIZE
00862     fprintf (stderr, "optimize_climbing_sequential: end\n");
00863 #endif
00864 }
00865
00866 static void *
00867 optimize_climbing_thread (ParallelData * data)
00868 {
00869     unsigned int i, thread;
00870     double e;
00871 #if DEBUG_OPTIMIZE
00872     fprintf (stderr, "optimize_climbing_thread: start\n");
00873 #endif
00874     thread = data->thread;
00875 #if DEBUG_OPTIMIZE
00876     fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
00877             thread,
00878             optimize->thread_climbing[thread],
00879             optimize->thread_climbing[thread + 1]);
00880 #endif
00881     for (i = optimize->thread_climbing[thread];
00882          i < optimize->thread_climbing[thread + 1]; ++i)
00883     {
00884         e = optimize_norm (i);
00885         g_mutex_lock (mutex);
00886         optimize_best_climbing (i, e);
00887         optimize_save_variables (i, e);
00888         if (e < optimize->threshold)
00889         {
00890             optimize->stop = 1;
00891             g_mutex_unlock (mutex);
00892             if (optimize->stop)
00893                 break;
00894 #if DEBUG_OPTIMIZE
00895             fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
00896 #endif
00897         }
00898 #if DEBUG_OPTIMIZE
00899     fprintf (stderr, "optimize_climbing_thread: end\n");
00900 #endif
00901     g_thread_exit (NULL);
00902     return NULL;
00903 }
00904
00905 static double
00906 optimize_estimate_climbing_random (unsigned int variable,
00907                                    unsigned int estimate
00908                                    __attribute__ ((unused)))
00909 {
00910     double x;
00911 #if DEBUG_OPTIMIZE
00912     fprintf (stderr, "optimize_estimate_climbing_random: start\n");
00913 #endif
00914     x = optimize->climbing[variable]
00915         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00916 #if DEBUG_OPTIMIZE
00917     fprintf (stderr, "optimize_estimate_climbing_random: climbing=%u=%lg\n",
00918             variable, x);
00919 #endif

```

```

00931     fprintf (stderr, "optimize_estimate_climbing_random: end\n");
00932 #endif
00933     return x;
00934 }
00935
00939 static double
00940 optimize_estimate_climbing_coordinates (unsigned int variable,
00941                                         unsigned int estimate)
00942 {
00943     double x;
00944     #if DEBUG_OPTIMIZE
00945     fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
00946 #endif
00947     x = optimize->climbing[variable];
00948     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00949     {
00950         if (estimate & 1)
00951             x += optimize->step[variable];
00952         else
00953             x -= optimize->step[variable];
00954     }
00955     #if DEBUG_OPTIMIZE
00956     fprintf (stderr,
00957             "optimize_estimate_climbing_coordinates: climbing=%u\n",
00958             variable, x);
00959     fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
00960 #endif
00961     return x;
00962 }
00963
00969 static inline void
00970 optimize_step_climbing (unsigned int simulation)
00971 {
00972     GThread *thread[nthreads_climbing];
00973     ParallelData data[nthreads_climbing];
00974     unsigned int i, j, k, b;
00975     #if DEBUG_OPTIMIZE
00976     fprintf (stderr, "optimize_step_climbing: start\n");
00977 #endif
00978     for (i = 0; i < optimize->nestimates; ++i)
00979     {
00980         k = (simulation + i) * optimize->nvariables;
00981         b = optimize->simulation_best[0] * optimize->nvariables;
00982         #if DEBUG_OPTIMIZE
00983         fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
00984                 simulation + i, optimize->simulation_best[0]);
00985         #endif
00986         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00987         {
00988             #if DEBUG_OPTIMIZE
00989             fprintf (stderr,
00990                     "optimize_step_climbing: estimate=%u best=%u\n",
00991                     i, j, optimize->value[b]);
00992             #endif
00993             optimize->value[k]
00994                 = optimize->value[b] + optimize_estimate_climbing (j, i);
00995             optimize->value[k] = fmin (fmax (optimize->value[k],
00996                                             optimize->rangeminabs[j]),
00997                                     optimize->rangemaxabs[j]);
00998             #if DEBUG_OPTIMIZE
00999             fprintf (stderr,
01000                     "optimize_step_climbing: estimate=%u variable=%u\n",
01001                     i, j, optimize->value[k]);
01002             #endif
01003         }
01004     }
01005     if (nthreads_climbing == 1)
01006         optimize_climbing_sequential (simulation);
01007     else
01008     {
01009         for (i = 0; i <= nthreads_climbing; ++i)
01010         {
01011             optimize->thread_climbing[i]
01012                 = simulation + optimize->nstart_climbing
01013                 + i * (optimize->nend_climbing - optimize->
01014                       nstart_climbing)
01015                 / nthreads_climbing;
01016             #if DEBUG_OPTIMIZE
01017             fprintf (stderr,
01018                     "optimize_step_climbing: i=%u thread_climbing=%u\n",
01019                     i, optimize->thread_climbing[i]);
01020             #endif
01021         }
01022         for (i = 0; i < nthreads_climbing; ++i)
01023         {
01024             data[i].thread = i;
01025             thread[i] = g_thread_new

```



```

01025         (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
01026     }
01027     for (i = 0; i < nthreads_climbing; ++i)
01028         g_thread_join (thread[i]);
01029 }
01030 #if DEBUG_OPTIMIZE
01031 fprintf (stderr, "optimize_step_climbing: end\n");
01032 #endif
01033 }
01034
01035 static inline void
01036 optimize_climbing ()
01037 {
01038     unsigned int i, j, k, b, s, adjust;
01039     #if DEBUG_OPTIMIZE
01040     fprintf (stderr, "optimize_climbing: start\n");
01041     #endif
01042     for (i = 0; i < optimize->nvariables; ++i)
01043         optimize->climbing[i] = 0.;
01044     b = optimize->simulation_best[0] * optimize->nvariables;
01045     s = optimize->nsimulations;
01046     adjust = 1;
01047     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01048     {
01049         #if DEBUG_OPTIMIZE
01050         fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
01051             i, optimize->simulation_best[0]);
01052         #endif
01053         optimize_step_climbing (s);
01054         k = optimize->simulation_best[0] * optimize->nvariables;
01055         #if DEBUG_OPTIMIZE
01056         fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
01057             i, optimize->simulation_best[0]);
01058         #endif
01059         if (k == b)
01060         {
01061             if (adjust)
01062             {
01063                 for (j = 0; j < optimize->nvariables; ++j)
01064                     optimize->step[j] *= 0.5;
01065                 for (j = 0; j < optimize->nvariables; ++j)
01066                     optimize->climbing[j] = 0.;
01067                 adjust = 1;
01068             }
01069             else
01070             {
01071                 for (j = 0; j < optimize->nvariables; ++j)
01072                 {
01073                     #if DEBUG_OPTIMIZE
01074                     fprintf (stderr,
01075                         "optimize_climbing: best%u=%%.14le old%u=%%.14le\n",
01076                         j, optimize->value[k + j], j, optimize->value[b + j]);
01077                     #endif
01078                     optimize->climbing[j]
01079                         = (1. - optimize->relaxation) * optimize->climbing[j]
01080                         + optimize->relaxation
01081                         * (optimize->value[k + j] - optimize->value[b + j]);
01082                     #if DEBUG_OPTIMIZE
01083                     fprintf (stderr, "optimize_climbing: climbing%u=%%.14le\n",
01084                         j, optimize->climbing[j]);
01085                     #endif
01086                 }
01087                 adjust = 0;
01088             }
01089         }
01090     }
01091     #if DEBUG_OPTIMIZE
01092     fprintf (stderr, "optimize_climbing: end\n");
01093     #endif
01094 }
01095
01096 static double
01097 optimize_genetic_objective (Entity * entity)
01098 {
01099     unsigned int j;
01100     double objective;
01101     char buffer[64];
01102     #if DEBUG_OPTIMIZE
01103     fprintf (stderr, "optimize_genetic_objective: start\n");
01104     #endif
01105     for (j = 0; j < optimize->nvariables; ++j)
01106     {
01107         optimize->value[entity->id * optimize->nvariables + j]
01108             = genetic_get_variable (entity, optimize->genetic_variable + j);
01109     }
01110     objective = optimize_norm (entity->id);
01111     g_mutex_lock (mutex);
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {

```

```

01120     snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01121     fprintf (optimize->file_variables, buffer,
01122             genetic_get_variable (entity, optimize->genetic_variable + j));
01123 }
01124 fprintf (optimize->file_variables, "%.14le\n", objective);
01125 g_mutex_unlock (mutex);
01126 #if DEBUG_OPTIMIZE
01127 fprintf (stderr, "optimize_genetic_objective: end\n");
01128 #endif
01129 return objective;
01130 }
01131
01132 static void
01133 optimize_genetic ()
01134 {
01135     double *best_variable = NULL;
01136     char *best_genome = NULL;
01137     double best_objective = 0.;
01138 #if DEBUG_OPTIMIZE
01139     fprintf (stderr, "optimize_genetic: start\n");
01140     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01141             nthreads);
01142     fprintf (stderr,
01143             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01144             optimize->nvariables, optimize->nsimulations, optimize->
01145             niterations);
01146     fprintf (stderr,
01147             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01148             optimize->mutation_ratio, optimize->reproduction_ratio,
01149             optimize->adaptation_ratio);
01150 #endif
01151     genetic_algorithm_default (optimize->nvariables,
01152                               optimize->genetic_variable,
01153                               optimize->nsimulations,
01154                               optimize->niterations,
01155                               optimize->mutation_ratio,
01156                               optimize->reproduction_ratio,
01157                               optimize->adaptation_ratio,
01158                               optimize->seed,
01159                               optimize->threshold,
01160                               &optimize_genetic_objective,
01161                               &best_genome, &best_variable, &best_objective);
01162 #if DEBUG_OPTIMIZE
01163     fprintf (stderr, "optimize_genetic: the best\n");
01164 #endif
01165     optimize->error_old = (double *) g_malloc (sizeof (double));
01166     optimize->value_old
01167         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01168     optimize->error_old[0] = best_objective;
01169     memcpy (optimize->value_old, best_variable,
01170            optimize->nvariables * sizeof (double));
01171     g_free (best_genome);
01172     g_free (best_variable);
01173     optimize_print ();
01174 #if DEBUG_OPTIMIZE
01175     fprintf (stderr, "optimize_genetic: end\n");
01176 #endif
01177 }
01178
01179 static inline void
01180 optimize_save_old ()
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         memcpy (optimize->value_old + i * optimize->nvariables,
01196                optimize->value + j * optimize->nvariables,
01197                optimize->nvariables * sizeof (double));
01198     }
01199 #if DEBUG_OPTIMIZE
01200     for (i = 0; i < optimize->nvariables; ++i)
01201         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01202                 i, optimize->value_old[i]);
01203     fprintf (stderr, "optimize_save_old: end\n");
01204 #endif
01205 }
01206
01207
01208
01209
01210
01211

```

```

01216 static inline void
01217 optimize_merge_old ()
01218 {
01219     unsigned int i, j, k;
01220     double v[optimize->nbest * optimize->nvariables], e[optimize->
nbest],
01221         *enew, *eold;
01222     #if DEBUG_OPTIMIZE
01223     fprintf (stderr, "optimize_merge_old: start\n");
01224     #endif
01225     enew = optimize->error_best;
01226     eold = optimize->error_old;
01227     i = j = k = 0;
01228     do
01229     {
01230         if (*enew < *eold)
01231         {
01232             memcpy (v + k * optimize->nvariables,
01233                     optimize->value
01234                     + optimize->simulation_best[i] * optimize->
nvariables,
01235                     optimize->nvariables * sizeof (double));
01236             e[k] = *enew;
01237             ++k;
01238             ++enew;
01239             ++i;
01240         }
01241         else
01242         {
01243             memcpy (v + k * optimize->nvariables,
01244                     optimize->value_old + j * optimize->nvariables,
01245                     optimize->nvariables * sizeof (double));
01246             e[k] = *eold;
01247             ++k;
01248             ++eold;
01249             ++j;
01250         }
01251     }
01252     while (k < optimize->nbest);
01253     memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01254     memcpy (optimize->error_old, e, k * sizeof (double));
01255     #if DEBUG_OPTIMIZE
01256     fprintf (stderr, "optimize_merge_old: end\n");
01257     #endif
01258 }
01259
01264 static inline void
01265 optimize_refine ()
01266 {
01267     unsigned int i, j;
01268     double d;
01269     #if HAVE_MPI
01270     MPI_Status mpi_stat;
01271     #endif
01272     #if DEBUG_OPTIMIZE
01273     fprintf (stderr, "optimize_refine: start\n");
01274     #endif
01275     #if HAVE_MPI
01276     if (!optimize->mpi_rank)
01277     {
01278     #endif
01279         for (j = 0; j < optimize->nvariables; ++j)
01280         {
01281             optimize->rangemin[j] = optimize->rangemax[j]
= optimize->value_old[j];
01282         }
01283         for (i = 0; ++i < optimize->nbest;)
01284         {
01285             for (j = 0; j < optimize->nvariables; ++j)
01286             {
01287                 optimize->rangemin[j]
= fmin (optimize->rangemin[j],
01288         optimize->value_old[i * optimize->nvariables + j]);
01289                 optimize->rangemax[j]
= fmax (optimize->rangemax[j],
01290         optimize->value_old[i * optimize->nvariables + j]);
01291             }
01292         }
01293         for (j = 0; j < optimize->nvariables; ++j)
01294         {
01295             d = optimize->tolerance
* (optimize->rangemax[j] - optimize->rangemin[j]);
01296             switch (optimize->algorithm)
01297             {
01298             case ALGORITHM_MONTE_CARLO:
01299                 d *= 0.5;
01300                 break;
01301             }
01302         }
01303     }
01304 }

```

```

01305         default:
01306             if (optimize->nsweeps[j] > 1)
01307                 d /= optimize->nsweeps[j] - 1;
01308             else
01309                 d = 0.;
01310         }
01311         optimize->rangemin[j] -= d;
01312         optimize->rangemin[j]
01313             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01314         optimize->rangemax[j] += d;
01315         optimize->rangemax[j]
01316             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01317         printf ("%s min=%lg max=%lg\n", optimize->label[j],
01318             optimize->rangemin[j], optimize->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; (int) i < ntasks; ++i)
01325     {
01326         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01327             1, MPI_COMM_WORLD);
01328         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01329             1, MPI_COMM_WORLD);
01330     }
01331 }
01332 else
01333 {
01334     MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01335         MPI_COMM_WORLD, &mpi_stat);
01336     MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01337         MPI_COMM_WORLD, &mpi_stat);
01338 }
01339 #endif
01340 #if DEBUG_OPTIMIZE
01341     fprintf (stderr, "optimize_refine: end\n");
01342 #endif
01343 }
01344
01345 static void
01346 optimize_step ()
01347 {
01348     #if DEBUG_OPTIMIZE
01349         fprintf (stderr, "optimize_step: start\n");
01350     #endif
01351     optimize_algorithm ();
01352     if (optimize->nsteps)
01353         optimize_climbing ();
01354     #if DEBUG_OPTIMIZE
01355         fprintf (stderr, "optimize_step: end\n");
01356     #endif
01357 }
01358
01359 static inline void
01360 optimize_iterate ()
01361 {
01362     unsigned int i;
01363     #if DEBUG_OPTIMIZE
01364         fprintf (stderr, "optimize_iterate: start\n");
01365     #endif
01366     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01367     optimize->value_old =
01368         (double *) g_malloc (optimize->nbest * optimize->nvariables *
01369             sizeof (double));
01370     optimize_step ();
01371     optimize_save_old ();
01372     optimize_refine ();
01373     optimize_print ();
01374     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01375     {
01376         optimize_step ();
01377         optimize_merge_old ();
01378         optimize_refine ();
01379         optimize_print ();
01380     }
01381     #if DEBUG_OPTIMIZE
01382         fprintf (stderr, "optimize_iterate: end\n");
01383     #endif
01384 }
01385
01386 void
01387 optimize_free ()
01388 {
01389     unsigned int i, j;
01390     #if DEBUG_OPTIMIZE
01391         fprintf (stderr, "optimize_free: start\n");
01392     #endif

```

```

01401 #endif
01402     for (j = 0; j < optimize->ninputs; ++j)
01403     {
01404         for (i = 0; i < optimize->nexperiments; ++i)
01405             g_mapped_file_unref (optimize->file[j][i]);
01406         g_free (optimize->file[j]);
01407     }
01408     g_free (optimize->error_old);
01409     g_free (optimize->value_old);
01410     g_free (optimize->value);
01411     g_free (optimize->genetic_variable);
01412     #if DEBUG_OPTIMIZE
01413     fprintf (stderr, "optimize_free: end\n");
01414     #endif
01415 }
01416
01420 void
01421 optimize_open ()
01422 {
01423     GTimeZone *tz;
01424     GDateTime *t0, *t;
01425     unsigned int i, j;
01426
01427     #if DEBUG_OPTIMIZE
01428     char *buffer;
01429     fprintf (stderr, "optimize_open: start\n");
01430     #endif
01431
01432     // Getting initial time
01433     #if DEBUG_OPTIMIZE
01434     fprintf (stderr, "optimize_open: getting initial time\n");
01435     #endif
01436     tz = g_time_zone_new_utc ();
01437     t0 = g_date_time_new_now (tz);
01438
01439     // Obtaining and initing the pseudo-random numbers generator seed
01440     #if DEBUG_OPTIMIZE
01441     fprintf (stderr, "optimize_open: getting initial seed\n");
01442     #endif
01443     if (optimize->seed == DEFAULT_RANDOM_SEED)
01444         optimize->seed = input->seed;
01445     gsl_rng_set (optimize->rng, optimize->seed);
01446
01447     // Replacing the working directory
01448     #if DEBUG_OPTIMIZE
01449     fprintf (stderr, "optimize_open: replacing the working directory\n");
01450     #endif
01451     g_chdir (input->directory);
01452
01453     // Getting results file names
01454     optimize->result = input->result;
01455     optimize->variables = input->variables;
01456
01457     // Obtaining the simulator file
01458     optimize->simulator = input->simulator;
01459
01460     // Obtaining the evaluator file
01461     optimize->evaluator = input->evaluator;
01462
01463     // Reading the algorithm
01464     optimize->algorithm = input->algorithm;
01465     switch (optimize->algorithm)
01466     {
01467         case ALGORITHM_MONTE_CARLO:
01468             optimize_algorithm = optimize_MonteCarlo;
01469             break;
01470         case ALGORITHM_SWEEP:
01471             optimize_algorithm = optimize_sweep;
01472             break;
01473         case ALGORITHM_ORTHOGONAL:
01474             optimize_algorithm = optimize_orthogonal;
01475             break;
01476         default:
01477             optimize_algorithm = optimize_genetic;
01478             optimize->mutation_ratio = input->mutation_ratio;
01479             optimize->reproduction_ratio = input->
01480 reproduction_ratio;
01481             optimize->adaptation_ratio = input->adaptation_ratio;
01482     }
01483     optimize->nvariables = input->nvariables;
01484     optimize->nsimulations = input->nsimulations;
01485     optimize->niterations = input->niterations;
01486     optimize->nbest = input->nbest;
01487     optimize->tolerance = input->tolerance;
01488     optimize->nsteps = input->nsteps;
01489     optimize->nestimates = 0;
01490     optimize->threshold = input->threshold;

```

```

01490     optimize->stop = 0;
01491     if (input->nsteps)
01492     {
01493         optimize->relaxation = input->relaxation;
01494         switch (input->climbing)
01495         {
01496             case CLIMBING_METHOD_COORDINATES:
01497                 optimize->nestimates = 2 * optimize->nvariables;
01498                 optimize_estimate_climbing =
optimize_estimate_climbing_coordinates;
01499                 break;
01500             default:
01501                 optimize->nestimates = input->nestimates;
01502                 optimize_estimate_climbing =
optimize_estimate_climbing_random;
01503         }
01504     }
01505
01506     #if DEBUG_OPTIMIZE
01507     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01508     #endif
01509     optimize->simulation_best
01510     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01511     optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01512
01513     // Reading the experimental data
01514     #if DEBUG_OPTIMIZE
01515     buffer = g_get_current_dir ();
01516     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01517     g_free (buffer);
01518     #endif
01519     optimize->nexperiments = input->nexperiments;
01520     optimize->ninputs = input->experiment->ninputs;
01521     optimize->experiment
01522     = (char **) alloca (input->nexperiments * sizeof (char *));
01523     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01524     for (i = 0; i < input->experiment->ninputs; ++i)
01525         optimize->file[i] = (GMappedFile **)
01526         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01527     for (i = 0; i < input->nexperiments; ++i)
01528     {
01529         #if DEBUG_OPTIMIZE
01530         fprintf (stderr, "optimize_open: i=%u\n", i);
01531         #endif
01532         optimize->experiment[i] = input->experiment[i].
name;
01533         optimize->weight[i] = input->experiment[i].weight;
01534         #if DEBUG_OPTIMIZE
01535         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
optimize->experiment[i], optimize->weight[i]);
01536         #endif
01537         for (j = 0; j < input->experiment->ninputs; ++j)
01538         {
01539             #if DEBUG_OPTIMIZE
01540             fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01541             #endif
01542             optimize->file[j][i]
01543             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01544         }
01545     }
01546 }
01547
01548 // Reading the variables data
01549 #if DEBUG_OPTIMIZE
01550 fprintf (stderr, "optimize_open: reading variables\n");
01551 #endif
01552 optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01553 j = input->nvariables * sizeof (double);
01554 optimize->rangemin = (double *) alloca (j);
01555 optimize->rangeminabs = (double *) alloca (j);
01556 optimize->rangemax = (double *) alloca (j);
01557 optimize->rangemaxabs = (double *) alloca (j);
01558 optimize->step = (double *) alloca (j);
01559 j = input->nvariables * sizeof (unsigned int);
01560 optimize->precision = (unsigned int *) alloca (j);
01561 optimize->nsweeps = (unsigned int *) alloca (j);
01562 optimize->nbits = (unsigned int *) alloca (j);
01563 for (i = 0; i < input->nvariables; ++i)
01564 {
01565     optimize->label[i] = input->variable[i].name;
01566     optimize->rangemin[i] = input->variable[i].rangemin;
01567     optimize->rangeminabs[i] = input->variable[i].
rangeminabs;
01568     optimize->rangemax[i] = input->variable[i].rangemax;
01569     optimize->rangemaxabs[i] = input->variable[i].
rangemaxabs;
01570     optimize->precision[i] = input->variable[i].
precision;

```

```

01571     optimize->step[i] = input->variable[i].step;
01572     optimize->nsweeps[i] = input->variable[i].nsweeps;
01573     optimize->nbits[i] = input->variable[i].nbits;
01574 }
01575 if (input->algorithm == ALGORITHM_SWEEP
01576 || input->algorithm == ALGORITHM_ORTHOGONAL)
01577 {
01578     optimize->nsimulations = 1;
01579     for (i = 0; i < input->nvariables; ++i)
01580     {
01581         optimize->nsimulations *= optimize->nsweeps[i];
01582 #if DEBUG_OPTIMIZE
01583         fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01584             optimize->nsweeps[i], optimize->nsimulations);
01585 #endif
01586     }
01587 }
01588 if (optimize->nsteps)
01589     optimize->climbing
01590     = (double *) alloca (optimize->nvariables * sizeof (double));
01591
01592 // Setting error norm
01593 switch (input->norm)
01594 {
01595     case ERROR_NORM_EUCLIDIAN:
01596         optimize_norm = optimize_norm_euclidian;
01597         break;
01598     case ERROR_NORM_MAXIMUM:
01599         optimize_norm = optimize_norm_maximum;
01600         break;
01601     case ERROR_NORM_P:
01602         optimize_norm = optimize_norm_p;
01603         optimize->p = input->p;
01604         break;
01605     default:
01606         optimize_norm = optimize_norm_taxicab;
01607 }
01608
01609 // Allocating values
01610 #if DEBUG_OPTIMIZE
01611     fprintf (stderr, "optimize_open: allocating variables\n");
01612     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01613         optimize->nvariables, optimize->algorithm);
01614 #endif
01615 optimize->genetic_variable = NULL;
01616 if (optimize->algorithm == ALGORITHM_GENETIC)
01617 {
01618     optimize->genetic_variable = (GeneticVariable *)
01619         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01620     for (i = 0; i < optimize->nvariables; ++i)
01621     {
01622 #if DEBUG_OPTIMIZE
01623         fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01624             i, optimize->rangemin[i], optimize->rangemax[i],
01625             optimize->nbits[i]);
01626 #endif
01627         optimize->genetic_variable[i].minimum = optimize->
01628             rangemin[i];
01629         optimize->genetic_variable[i].maximum = optimize->
01630             rangemax[i];
01631         optimize->genetic_variable[i].nbits = optimize->nbits[i];
01632     }
01633 #if DEBUG_OPTIMIZE
01634     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01635         optimize->nvariables, optimize->nsimulations);
01636 #endif
01637     optimize->value = (double *)
01638         g_malloc ((optimize->nsimulations
01639             + optimize->nestimates * optimize->nsteps)
01640             * optimize->nvariables * sizeof (double));
01641
01642 // Calculating simulations to perform for each task
01643 #if HAVE_MPI
01644 #if DEBUG_OPTIMIZE
01645     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01646         optimize->mpi_rank, ntasks);
01647 #endif
01648     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01649         ntasks;
01650     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01651         ntasks;
01652     if (optimize->nsteps)
01653     {
01654         optimize->nstart_climbing
01655             = optimize->mpi_rank * optimize->nestimates / ntasks;
01656         optimize->nend_climbing

```

```

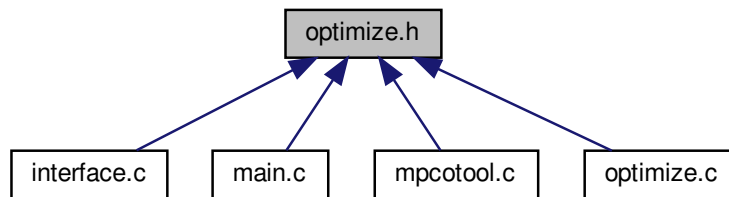
01654         = (1 + optimize->mpi_rank) * optimize->nestimates /
    ntasks;
01655     }
01656 #else
01657     optimize->nstart = 0;
01658     optimize->nend = optimize->nsimulations;
01659     if (optimize->nsteps)
01660     {
01661         optimize->nstart_climbing = 0;
01662         optimize->nend_climbing = optimize->nestimates;
01663     }
01664 #endif
01665 #if DEBUG_OPTIMIZE
01666     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01667             optimize->nend);
01668 #endif
01669
01670 // Calculating simulations to perform for each thread
01671 optimize->thread
01672     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01673 for (i = 0; i <= nthreads; ++i)
01674     {
01675         optimize->thread[i] = optimize->nstart
01676             + i * (optimize->nend - optimize->nstart) / nthreads;
01677 #if DEBUG_OPTIMIZE
01678         fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01679                 optimize->thread[i]);
01680 #endif
01681     }
01682     if (optimize->nsteps)
01683         optimize->thread_climbing = (unsigned int *)
01684             alloca ((1 + nthreads_climbing) * sizeof (unsigned int));
01685
01686 // Opening result files
01687 optimize->file_result = g_fopen (optimize->result, "w");
01688 optimize->file_variables = g_fopen (optimize->variables, "w");
01689
01690 // Performing the algorithm
01691 switch (optimize->algorithm)
01692     {
01693         // Genetic algorithm
01694         case ALGORITHM_GENETIC:
01695             optimize_genetic ();
01696             break;
01697
01698         // Iterative algorithm
01699         default:
01700             optimize_iterate ();
01701     }
01702
01703 // Getting calculation time
01704 t = g_date_time_new_now (tz);
01705 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01706 g_date_time_unref (t);
01707 g_date_time_unref (t0);
01708 g_time_zone_unref (tz);
01709 printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01710 fprintf (optimize->file_result, "%s = %.6lg s\n",
01711         _("Calculation time"), optimize->calculation_time);
01712
01713 // Closing result files
01714 fclose (optimize->file_variables);
01715 fclose (optimize->file_result);
01716
01717 #if DEBUG_OPTIMIZE
01718     fprintf (stderr, "optimize_open: end\n");
01719 #endif
01720 }

```

4.23 optimize.h File Reference

Header file to define the optimization functions.

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



Data Structures

- struct [Optimize](#)
Struct to define the optimization ation data.
- struct [ParallelData](#)
Struct to pass to the GThreads parallelized function.

Functions

- void [optimize_free](#) ()
- void [optimize_open](#) ()

Variables

- int [ntasks](#)
Tasks number.
- unsigned int [nthreads](#)
Threads number.
- unsigned int [nthreads_climbing](#)
Number of threads for the hill climbing method.
- GMutex [mutex](#) [1]
GMutex struct.
- [Optimize](#) [optimize](#) [1]
Optimization data.

4.23.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.23.2 Function Documentation

4.23.2.1 optimize_free()

```
void optimize_free ( )
```

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

Definition at line 1396 of file [optimize.c](#).

```
01397 {
01398     unsigned int i, j;
01399     #if DEBUG_OPTIMIZE
01400     fprintf (stderr, "optimize_free: start\n");
01401     #endif
01402     for (j = 0; j < optimize->ninputs; ++j)
01403     {
01404         for (i = 0; i < optimize->nexperiments; ++i)
01405             g_mapped_file_unref (optimize->file[j][i]);
01406         g_free (optimize->file[j]);
01407     }
01408     g_free (optimize->error_old);
01409     g_free (optimize->value_old);
01410     g_free (optimize->value);
01411     g_free (optimize->genetic_variable);
01412     #if DEBUG_OPTIMIZE
01413     fprintf (stderr, "optimize_free: end\n");
01414     #endif
01415 }
```

4.23.2.2 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1421 of file [optimize.c](#).

```
01422 {
01423     GTimeZone *tz;
01424     GDateTime *t0, *t;
01425     unsigned int i, j;
01426
01427     #if DEBUG_OPTIMIZE
01428     char *buffer;
01429     fprintf (stderr, "optimize_open: start\n");
01430     #endif
01431
01432     // Getting initial time
01433     #if DEBUG_OPTIMIZE
01434     fprintf (stderr, "optimize_open: getting initial time\n");
01435     #endif
01436     tz = g_time_zone_new_utc ();
01437     t0 = g_date_time_new_now (tz);
01438
01439     // Obtaining and initing the pseudo-random numbers generator seed
01440     #if DEBUG_OPTIMIZE
01441     fprintf (stderr, "optimize_open: getting initial seed\n");
01442     #endif
01443     if (optimize->seed == DEFAULT_RANDOM_SEED)
01444         optimize->seed = input->seed;
01445     gsl_rng_set (optimize->rng, optimize->seed);
01446 }
```

```

01447 // Replacing the working directory
01448 #if DEBUG_OPTIMIZE
01449     fprintf (stderr, "optimize_open: replacing the working directory\n");
01450 #endif
01451     g_chdir (input->directory);
01452
01453 // Getting results file names
01454     optimize->result = input->result;
01455     optimize->variables = input->variables;
01456
01457 // Obtaining the simulator file
01458     optimize->simulator = input->simulator;
01459
01460 // Obtaining the evaluator file
01461     optimize->evaluator = input->evaluator;
01462
01463 // Reading the algorithm
01464     optimize->algorithm = input->algorithm;
01465     switch (optimize->algorithm)
01466     {
01467     case ALGORITHM_MONTE_CARLO:
01468         optimize_algorithm = optimize_MonteCarlo;
01469         break;
01470     case ALGORITHM_SWEEP:
01471         optimize_algorithm = optimize_sweep;
01472         break;
01473     case ALGORITHM_ORTHOGONAL:
01474         optimize_algorithm = optimize_orthogonal;
01475         break;
01476     default:
01477         optimize_algorithm = optimize_genetic;
01478         optimize->mutation_ratio = input->
mutation_ratio;
01479         optimize->reproduction_ratio = input->
reproduction_ratio;
01480         optimize->adaptation_ratio = input->
adaptation_ratio;
01481     }
01482     optimize->nvariables = input->nvariables;
01483     optimize->nsimulations = input->nsimulations;
01484     optimize->niterations = input->niterations;
01485     optimize->nbest = input->nbest;
01486     optimize->tolerance = input->tolerance;
01487     optimize->nsteps = input->nsteps;
01488     optimize->nestimates = 0;
01489     optimize->threshold = input->threshold;
01490     optimize->stop = 0;
01491     if (input->nsteps)
01492     {
01493         optimize->relaxation = input->relaxation;
01494         switch (input->climbing)
01495         {
01496         case CLIMBING_METHOD_COORDINATES:
01497             optimize->nestimates = 2 * optimize->
nvariables;
01498             optimize_estimate_climbing =
optimize_estimate_climbing_coordinates;
01499             break;
01500         default:
01501             optimize->nestimates = input->nestimates;
01502             optimize_estimate_climbing =
optimize_estimate_climbing_random;
01503         }
01504     }
01505
01506 #if DEBUG_OPTIMIZE
01507     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01508 #endif
01509     optimize->simulation_best
01510     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01511     optimize->error_best = (double *) alloca (optimize->
nbest * sizeof (double));
01512
01513 // Reading the experimental data
01514 #if DEBUG_OPTIMIZE
01515     buffer = g_get_current_dir ();
01516     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01517     g_free (buffer);
01518 #endif
01519     optimize->nexperiments = input->nexperiments;
01520     optimize->ninputs = input->experiment->ninputs;
01521     optimize->experiment
01522     = (char **) alloca (input->nexperiments * sizeof (char *));
01523     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double
));
01524     for (i = 0; i < input->experiment->ninputs; ++i)
01525         optimize->file[i] = (GMappedFile **)

```

```

01526     g_malloc (input->nexperiments * sizeof (GMappedFile *));
01527     for (i = 0; i < input->nexperiments; ++i)
01528     {
01529 #if DEBUG_OPTIMIZE
01530         fprintf (stderr, "optimize_open: i=%u\n", i);
01531 #endif
01532         optimize->experiment[i] = input->experiment[i].
01533             name;
01534         optimize->weight[i] = input->experiment[i].
01535             weight;
01536 #if DEBUG_OPTIMIZE
01537         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01538             optimize->experiment[i], optimize->
01539             weight[i]);
01540 #endif
01541         for (j = 0; j < input->experiment->ninputs; ++j)
01542         {
01543 #if DEBUG_OPTIMIZE
01544             fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01545 #endif
01546             optimize->file[j][i]
01547                 = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01548         }
01549     }
01550 // Reading the variables data
01551 #if DEBUG_OPTIMIZE
01552     fprintf (stderr, "optimize_open: reading variables\n");
01553 #endif
01554 optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01555 j = input->nvariables * sizeof (double);
01556 optimize->rangemin = (double *) alloca (j);
01557 optimize->rangeminabs = (double *) alloca (j);
01558 optimize->rangemax = (double *) alloca (j);
01559 optimize->rangemaxabs = (double *) alloca (j);
01560 optimize->step = (double *) alloca (j);
01561 j = input->nvariables * sizeof (unsigned int);
01562 optimize->precision = (unsigned int *) alloca (j);
01563 optimize->nsweeps = (unsigned int *) alloca (j);
01564 optimize->nbits = (unsigned int *) alloca (j);
01565 for (i = 0; i < input->nvariables; ++i)
01566 {
01567     optimize->label[i] = input->variable[i].name;
01568     optimize->rangemin[i] = input->variable[i].
01569         rangemin;
01570     optimize->rangeminabs[i] = input->variable[i].
01571         rangeminabs;
01572     optimize->rangemax[i] = input->variable[i].
01573         rangemax;
01574     optimize->rangemaxabs[i] = input->variable[i].
01575         rangemaxabs;
01576     optimize->precision[i] = input->variable[i].
01577         precision;
01578     optimize->step[i] = input->variable[i].step;
01579     optimize->nsweeps[i] = input->variable[i].
01580         nsweeps;
01581     optimize->nbits[i] = input->variable[i].nbits;
01582 }
01583 if (input->algorithm == ALGORITHM_SWEEP
01584     || input->algorithm == ALGORITHM_ORTHOGONAL)
01585 {
01586     optimize->nsimulations = 1;
01587     for (i = 0; i < input->nvariables; ++i)
01588     {
01589         optimize->nsimulations *= optimize->
01590             nsweeps[i];
01591 #if DEBUG_OPTIMIZE
01592         fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01593             optimize->nsweeps[i], optimize->
01594             nsimulations);
01595 #endif
01596     }
01597 }
01598 if (optimize->nsteps)
01599     optimize->climbing
01600         = (double *) alloca (optimize->nvariables * sizeof (double));
01601 // Setting error norm
01602 switch (input->norm)
01603 {
01604     case ERROR_NORM_EUCLIDIAN:
01605         optimize_norm = optimize_norm_euclidian;
01606         break;
01607     case ERROR_NORM_MAXIMUM:
01608         optimize_norm = optimize_norm_maximum;
01609         break;
01610     case ERROR_NORM_P:

```

```

01602         optimize_norm = optimize_norm_p;
01603         optimize->p = input->p;
01604         break;
01605     default:
01606         optimize_norm = optimize_norm_taxicab;
01607     }
01608
01609     // Allocating values
01610     #if DEBUG_OPTIMIZE
01611     fprintf (stderr, "optimize_open: allocating variables\n");
01612     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01613             optimize->nvariables, optimize->algorithm);
01614     #endif
01615     optimize->genetic_variable = NULL;
01616     if (optimize->algorithm == ALGORITHM_GENETIC)
01617     {
01618         optimize->genetic_variable = (GeneticVariable *)
01619         g_malloc (optimize->nvariables * sizeof (
01620         GeneticVariable));
01621         for (i = 0; i < optimize->nvariables; ++i)
01622         {
01623             #if DEBUG_OPTIMIZE
01624             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01625                     i, optimize->rangemin[i], optimize->
01626                     rangemax[i], optimize->nbits[i]);
01627             #endif
01628             optimize->genetic_variable[i].minimum =
01629             optimize->rangemin[i];
01630             optimize->genetic_variable[i].maximum =
01631             optimize->rangemax[i];
01632             optimize->genetic_variable[i].nbits = optimize->
01633             nbits[i];
01634         }
01635     }
01636     #if DEBUG_OPTIMIZE
01637     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01638             optimize->nvariables, optimize->
01639             nsimulations);
01640     #endif
01641     optimize->value = (double *)
01642     g_malloc ((optimize->nsimulations
01643             + optimize->nestimates * optimize->
01644             nsteps)
01645             * optimize->nvariables * sizeof (double));
01646
01647     // Calculating simulations to perform for each task
01648     #if HAVE_MPI
01649     #if DEBUG_OPTIMIZE
01650     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01651             optimize->mpi_rank, ntasks);
01652     #endif
01653     optimize->nstart = optimize->mpi_rank * optimize->
01654     nsimulations / ntasks;
01655     optimize->nend = (1 + optimize->mpi_rank) *
01656     optimize->nsimulations / ntasks;
01657     if (optimize->nsteps)
01658     {
01659         optimize->nstart_climbing
01660         = optimize->mpi_rank * optimize->nestimates /
01661         ntasks;
01662         optimize->nend_climbing
01663         = (1 + optimize->mpi_rank) * optimize->
01664         nestimates / ntasks;
01665     }
01666     #else
01667     optimize->nstart = 0;
01668     optimize->nend = optimize->nsimulations;
01669     if (optimize->nsteps)
01670     {
01671         optimize->nstart_climbing = 0;
01672         optimize->nend_climbing = optimize->
01673         nestimates;
01674     }
01675     #endif
01676     #if DEBUG_OPTIMIZE
01677     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->
01678             nstart,
01679             optimize->nend);
01680     #endif
01681
01682     // Calculating simulations to perform for each thread
01683     optimize->thread
01684     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01685     for (i = 0; i <= nthreads; ++i)
01686     {
01687         optimize->thread[i] = optimize->nstart

```



```

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;
00068     double adaptation_ratio;
00069     double relaxation;
00070     double calculation_time;
00071     double p;
00072     double threshold;
00073     unsigned long int seed;
00074     unsigned int nvariables;
00075     unsigned int nexperiments;
00076     unsigned int ninputs;
00077     unsigned int nsimulations;
00078     unsigned int nsteps;
00079     unsigned int nestimates;
00080     unsigned int algorithm;
00081     unsigned int nstart;
00082     unsigned int nend;
00083     unsigned int nstart_climbing;
00084     unsigned int nend_climbing;
00085     unsigned int niterations;
00086     unsigned int nbest;
00087     unsigned int nsaveds;
00088     unsigned int stop;
00089 #if HAVE_MPI

```

```

00113     int mpi_rank;
00114 #endif
00115 } Optimize;
00116
00121 typedef struct
00122 {
00123     unsigned int thread;
00124 } ParallelData;
00125
00126 // Global variables
00127 extern int ntasks;
00128 extern unsigned int nthreads;
00129 extern unsigned int nthreads_climbing;
00130 extern GMutex mutex[1];
00131 extern Optimize optimize[1];
00132
00133 // Public functions
00134 void optimize_free ();
00135 void optimize_open ();
00136
00137 #endif

```

4.25 utils.c File Reference

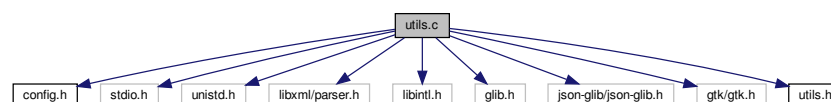
Source file to define some useful functions.

```

#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.

4.25.1 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](#).

4.25.2 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.

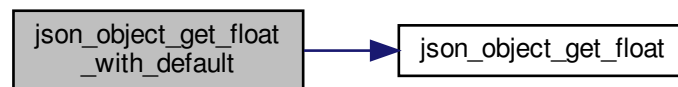
Definition at line 376 of file [utils.c](#).

```

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.

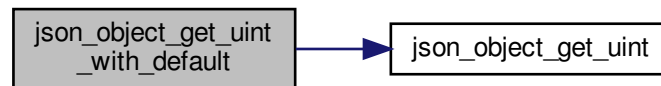
Parameters

<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.

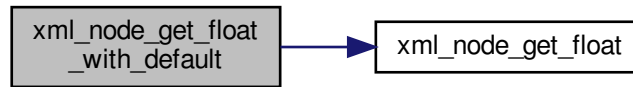
Parameters

<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.

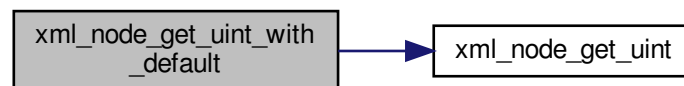
Parameters

<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.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     __attribute__ ((unused))
00061 #endif
00062 {
00063     #if HAVE_GTK
00064         GtkMessageDialog *dlg;
00065
00066         // Creating the dialog
00067         dlg = (GtkMessageDialog *)
00068             gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00069                                   (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00070
00071         // Setting the dialog title
00072         gtk_window_set_title (GTK_WINDOW (dlg), title);
00073
00074         // Showing the dialog and waiting response
00075         gtk_dialog_run (GTK_DIALOG (dlg));
00076     #endif
00077 }

```

```
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 }
00096
00100 void
00101 show_error (char *msg)
00102 {
00103     show_message (_("ERROR!"), msg, ERROR_TYPE);
00104 }
00105
00111 int
00112 xml_node_get_int (xmlNode * node,
00113                  const xmlChar * prop,
00114                  int *error_code)
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 }
00131
00137 unsigned int
00138 xml_node_get_uint (xmlNode * node,
00139                   const xmlChar * prop,
00140                   int *error_code)
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 }
00157
00164 unsigned int
00165 xml_node_get_uint_with_default (xmlNode * node,
00166                                const xmlChar * prop,
00167                                unsigned int default_value,
00168                                int *error_code)
00169 {
00170     unsigned int i;
00171     if (xmlHasProp (node, prop))
00172         i = xml_node_get_uint (node, prop, error_code);
00173     else
00174     {
00175         i = default_value;
00176         *error_code = 0;
00177     }
00178     return i;
00179 }
00180
00181
00187 double
00188 xml_node_get_float (xmlNode * node,
00189                    const xmlChar * prop,
00190                    int *error_code)
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 }
00207
00214 double
00215 xml_node_get_float_with_default (xmlNode * node,
00216                                 const xmlChar * prop,
00217                                 double default_value,
00218                                 int *error_code)
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 }
00230
00234 void
00235 xml_node_set_int (xmlNode * node,
00236                  const xmlChar * prop,
00237                  int value)
00238 {
00239     xmlChar buffer[64];
00240     snprintf ((char *) buffer, 64, "%d", value);
00241     xmlSetProp (node, prop, buffer);
00242 }
00243
00247 void
00248 xml_node_set_uint (xmlNode * node,
00249                   const xmlChar * prop,
00250                   unsigned int value)
00251 {
00252     xmlChar buffer[64];
00253     snprintf ((char *) buffer, 64, "%u", value);
00254     xmlSetProp (node, prop, buffer);
00255 }
00256
00260 void
00261 xml_node_set_float (xmlNode * node,
00262                    const xmlChar * prop,
00263                    double value)
00264 {
00265     xmlChar buffer[64];
00266     snprintf ((char *) buffer, 64, "%.14lg", value);
00267     xmlSetProp (node, prop, buffer);
00268 }
00269
00275 int
00276 json_object_get_int (JsonObject * object,
00277                     const char *prop,
00278                     int *error_code)
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 }
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     unsigned int i;
00333     if (json_object_get_member (object, prop))
00334         i = json_object_get_uint (object, prop, error_code);
00335     else
00336     {
00337         i = default_value;
00338         *error_code = 0;
00339     }
00340     return i;
00341 }
00342
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     double x;
00382     if (json_object_get_member (object, prop))
00383         x = json_object_get_float (object, prop, error_code);
00384     else
00385     {
00386         x = default_value;
00387         *error_code = 0;
00388     }
00389     return x;
00390 }
00391
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 }
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 {

```

```

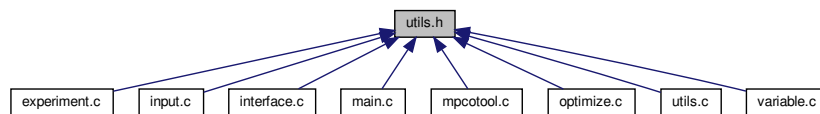
00429  char buffer[64];
00430  snprintf (buffer, 64, "%.14lg", value);
00431  json_object_set_string_member (object, prop, buffer);
00432 }
00433
00439  int
00440  cores_number ()
00441  {
00442  #ifdef G_OS_WIN32
00443      SYSTEM_INFO sysinfo;
00444      GetSystemInfo (&sysinfo);
00445      return 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

```

4.27 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.

4.27.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.27.2 Function Documentation

4.27.2.1 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 }
```

4.27.2.2 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 }
```

4.27.2.3 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 }
```

4.27.2.4 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.

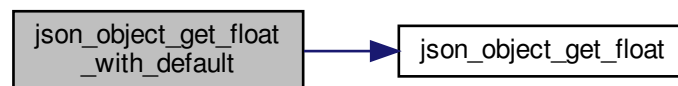
Parameters

<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:



4.27.2.5 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 }

```

4.27.2.6 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 }

```


4.27.2.7 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.

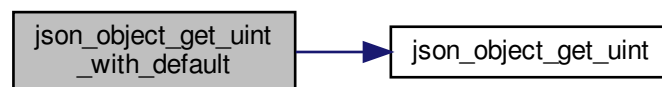
Parameters

<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:



4.27.2.8 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 }
```

4.27.2.9 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 }
```

4.27.2.10 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 }
```

4.27.2.11 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 }
```

4.27.2.12 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:



4.27.2.13 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 }
```

4.27.2.14 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 }
```

4.27.2.15 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.

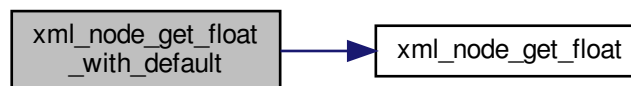
Parameters

<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:

**4.27.2.16 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 }
```

4.27.2.17 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 }
```

4.27.2.18 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.

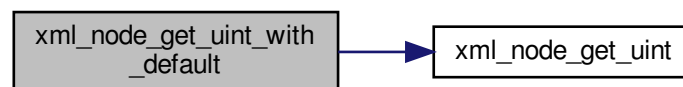
Parameters

<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:



4.27.2.19 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 }
```

4.27.2.20 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 }
```

4.27.2.21 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 }
```

4.28 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

```

4.29 variable.c File Reference

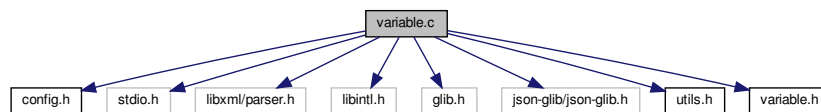
Source file to define the variable data.

```

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

```

Include dependency graph for variable.c:



Macros

- `#define DEBUG_VARIABLE 0`
Macro to debug variable functions.

Functions

- void `variable_free` (Variable *variable, unsigned int type)
- void `variable_error` (Variable *variable, char *message)
- int `variable_open_xml` (Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)
- int `variable_open_json` (Variable *variable, XmlNode *node, unsigned int algorithm, unsigned int nsteps)

Variables

- `const char * format [NPRECISIONS]`
Array of C-strings with variable formats.
- `const double precision [NPRECISIONS]`
Array of variable precisions.

4.29.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.29.2 Function Documentation

4.29.2.1 `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 85 of file [variable.c](#).

```
00089 {
00090     char buffer[64];
00091     if (!variable->name)
00092         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00093     else
00094         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00095     error\_message = g\_strdup (buffer);
00096 }
```

4.29.2.2 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 64 of file [variable.c](#).

```
00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_free: start\n");
00071     #endif
00072     if (type == INPUT_TYPE_XML)
00073         xmlFree (variable->name);
00074     else
00075         g_free (variable->name);
00076     #if DEBUG_VARIABLE
00077         fprintf (stderr, "variable_free: end\n");
00078     #endif
00079 }
```

4.29.2.3 variable_open_json()

```
int variable_open_json (
    Variable * variable,
    JsonNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

<i>variable</i>	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 264 of file [variable.c](#).

```
00269 {
```

```

00270     JsonObject *object;
00271     const char *label;
00272     int error_code;
00273     #if DEBUG_VARIABLE
00274     fprintf (stderr, "variable_open_json: start\n");
00275     #endif
00276     object = json_node_get_object (node);
00277     label = json_object_get_string_member (object, LABEL_NAME);
00278     if (!label)
00279     {
00280         variable_error (variable, _("no name"));
00281         goto exit_on_error;
00282     }
00283     variable->name = g_strdup (label);
00284     if (json_object_get_member (object, LABEL_MINIMUM))
00285     {
00286         variable->rangemin
00287         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00288         if (error_code)
00289         {
00290             variable_error (variable, _("bad minimum"));
00291             goto exit_on_error;
00292         }
00293         variable->rangeminabs
00294         = json_object_get_float_with_default (object,
00295         LABEL_ABSOLUTE_MINIMUM,
00296         -G_MAXDOUBLE, &error_code);
00297         if (error_code)
00298         {
00299             variable_error (variable, _("bad absolute minimum"));
00300             goto exit_on_error;
00301         }
00302         if (variable->rangemin < variable->rangeminabs)
00303         {
00304             variable_error (variable, _("minimum range not allowed"));
00305             goto exit_on_error;
00306         }
00307     }
00308     else
00309     {
00310         variable_error (variable, _("no minimum range"));
00311         goto exit_on_error;
00312     }
00313     if (json_object_get_member (object, LABEL_MAXIMUM))
00314     {
00315         variable->rangemax
00316         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00317         if (error_code)
00318         {
00319             variable_error (variable, _("bad maximum"));
00320             goto exit_on_error;
00321         }
00322         variable->rangemaxabs
00323         = json_object_get_float_with_default (object,
00324         LABEL_ABSOLUTE_MAXIMUM,
00325         G_MAXDOUBLE, &error_code);
00326         if (error_code)
00327         {
00328             variable_error (variable, _("bad absolute maximum"));
00329             goto exit_on_error;
00330         }
00331         if (variable->rangemax > variable->rangemaxabs)
00332         {
00333             variable_error (variable, _("maximum range not allowed"));
00334             goto exit_on_error;
00335         }
00336         if (variable->rangemax < variable->rangemin)
00337         {
00338             variable_error (variable, _("bad range"));
00339             goto exit_on_error;
00340         }
00341     }
00342     else
00343     {
00344         variable_error (variable, _("no maximum range"));
00345         goto exit_on_error;
00346     }
00347     variable->precision
00348     = json_object_get_uint_with_default (object,
00349     LABEL_PRECISION,
00350     DEFAULT_PRECISION, &error_code);
00351     if (error_code || variable->precision >= NPRECISIONS)
00352     {
00353         variable_error (variable, _("bad precision"));
00354         goto exit_on_error;
00355     }
00356     if (algorithm == ALGORITHM_SWEEP || algorithm ==

```

```

    ALGORITHM_ORTHOGONAL)
00354 {
00355     if (json_object_get_member (object, LABEL_NSWEEPS))
00356     {
00357         variable->nsweeps
00358         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00359         if (error_code || !variable->nsweeps)
00360         {
00361             variable_error (variable, _("bad sweeps"));
00362             goto exit_on_error;
00363         }
00364     }
00365     else
00366     {
00367         variable_error (variable, _("no sweeps number"));
00368         goto exit_on_error;
00369     }
00370 #if DEBUG_VARIABLE
00371     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00372 #endif
00373 }
00374 if (algorithm == ALGORITHM_GENETIC)
00375 {
00376     // Obtaining bits representing each variable
00377     if (json_object_get_member (object, LABEL_NBITS))
00378     {
00379         variable->nbits
00380         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00381         if (error_code || !variable->nbits)
00382         {
00383             variable_error (variable, _("invalid bits number"));
00384             goto exit_on_error;
00385         }
00386     }
00387     else
00388     {
00389         variable_error (variable, _("no bits number"));
00390         goto exit_on_error;
00391     }
00392 }
00393 else if (nsteps)
00394 {
00395     variable->step = json_object_get_float (object,
00396     LABEL_STEP, &error_code);
00397     if (error_code || variable->step < 0.)
00398     {
00399         variable_error (variable, _("bad step size"));
00400         goto exit_on_error;
00401     }
00402 }
00403 #if DEBUG_VARIABLE
00404     fprintf (stderr, "variable_open_json: end\n");
00405 #endif
00406     return 1;
00407 exit_on_error:
00408     variable_free (variable, INPUT_TYPE_JSON);
00409 #if DEBUG_VARIABLE
00410     fprintf (stderr, "variable_open_json: end\n");
00411 #endif
00412     return 0;
00413 }

```

Here is the call graph for this function:



4.29.2.4 `variable_open_xml()`

```
int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 104 of file [variable.c](#).

```
00109 {
00110     int error_code;
00111
00112     #if DEBUG_VARIABLE
00113         fprintf (stderr, "variable_open_xml: start\n");
00114     #endif
00115
00116     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00117     if (!variable->name)
00118     {
00119         variable_error (variable, _("no name"));
00120         goto exit_on_error;
00121     }
00122     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00123     {
00124         variable->rangemin
00125             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00126                                   &error_code);
00127         if (error_code)
00128         {
00129             variable_error (variable, _("bad minimum"));
00130             goto exit_on_error;
00131         }
00132         variable->rangeminabs = xml_node_get_float_with_default
00133             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00134             &error_code);
00135         if (error_code)
00136         {
00137             variable_error (variable, _("bad absolute minimum"));
00138             goto exit_on_error;
00139         }
00140         if (variable->rangemin < variable->rangeminabs)
00141         {
00142             variable_error (variable, _("minimum range not allowed"));
00143             goto exit_on_error;
00144         }
00145     }
00146     else
00147     {
00148         variable_error (variable, _("no minimum range"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00152     {
00153         variable->rangemax
00154             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00155                                   &error_code);
```



```

00156     if (error_code)
00157     {
00158         variable_error (variable, _("bad maximum"));
00159         goto exit_on_error;
00160     }
00161     variable->rangemaxabs = xml_node_get_float_with_default
00162     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00163      &error_code);
00164     if (error_code)
00165     {
00166         variable_error (variable, _("bad absolute maximum"));
00167         goto exit_on_error;
00168     }
00169     if (variable->rangemax > variable->rangemaxabs)
00170     {
00171         variable_error (variable, _("maximum range not allowed"));
00172         goto exit_on_error;
00173     }
00174     if (variable->rangemax < variable->rangemin)
00175     {
00176         variable_error (variable, _("bad range"));
00177         goto exit_on_error;
00178     }
00179 }
00180 else
00181 {
00182     variable_error (variable, _("no maximum range"));
00183     goto exit_on_error;
00184 }
00185 variable->precision
00186 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                DEFAULT_PRECISION, &error_code);
00187 if (error_code || variable->precision >= NPRECISIONS)
00188 {
00189     variable_error (variable, _("bad precision"));
00190     goto exit_on_error;
00191 }
00192 }
00193 if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
00194 {
00195     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00196     {
00197         variable->nsweeps
00198         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
                                &error_code);
00199         if (error_code || !variable->nsweeps)
00200         {
00201             variable_error (variable, _("bad sweeps"));
00202             goto exit_on_error;
00203         }
00204     }
00205 }
00206 else
00207 {
00208     variable_error (variable, _("no sweeps number"));
00209     goto exit_on_error;
00210 }
00211 #if DEBUG_VARIABLE
00212 fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00213 #endif
00214 }
00215 if (algorithm == ALGORITHM_GENETIC)
00216 {
00217     // Obtaining bits representing each variable
00218     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00219     {
00220         variable->nbits
00221         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                &error_code);
00222         if (error_code || !variable->nbits)
00223         {
00224             variable_error (variable, _("invalid bits number"));
00225             goto exit_on_error;
00226         }
00227     }
00228 }
00229 else
00230 {
00231     variable_error (variable, _("no bits number"));
00232     goto exit_on_error;
00233 }
00234 }
00235 else if (nsteps)
00236 {
00237     variable->step
00238     = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
00239     if (error_code || variable->step < 0.)

```

```

00240     {
00241         variable_error (variable, _("bad step size"));
00242         goto exit_on_error;
00243     }
00244 }
00245
00246 #if DEBUG_VARIABLE
00247 fprintf (stderr, "variable_open_xml: end\n");
00248 #endif
00249 return 1;
00250 exit_on_error:
00251 variable_free (variable, INPUT_TYPE_XML);
00252 #if DEBUG_VARIABLE
00253 fprintf (stderr, "variable_open_xml: end\n");
00254 #endif
00255 return 0;
00256 }

```

Here is the call graph for this function:



4.29.3 Variable Documentation

4.29.3.1 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](#).

4.29.3.2 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](#).

4.30 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     "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00046     "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
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_free (Variable * variable,
00056               unsigned int type)
00057 {
00058     #if DEBUG_VARIABLE
00059         fprintf (stderr, "variable_free: start\n");
00060     #endif
00061     if (type == INPUT_TYPE_XML)
00062         xmlFree (variable->name);
00063     else
00064         g_free (variable->name);
00065     #if DEBUG_VARIABLE
00066         fprintf (stderr, "variable_free: end\n");
00067     #endif
00068 }
00069
00070 void
00071 variable_error (Variable * variable,
00072                char *message)
00073 {
00074     char buffer[64];
00075     if (!variable->name)
00076         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00077     else
00078         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00079     error_message = g_strdup (buffer);
00080 }
00081
00082 int
00083 variable_open_xml (Variable * variable,
00084                   xmlNode * node,

```

```

00106             unsigned int algorithm,
00107             unsigned int nsteps)
00109 {
00110     int error_code;
00111
00112     #if DEBUG_VARIABLE
00113         fprintf (stderr, "variable_open_xml: start\n");
00114     #endif
00115
00116     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00117     if (!variable->name)
00118     {
00119         variable_error (variable, _("no name"));
00120         goto exit_on_error;
00121     }
00122     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00123     {
00124         variable->rangemin
00125             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00126                                   &error_code);
00127         if (error_code)
00128         {
00129             variable_error (variable, _("bad minimum"));
00130             goto exit_on_error;
00131         }
00132         variable->rangeminabs = xml_node_get_float_with_default
00133             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00134             &error_code);
00135         if (error_code)
00136         {
00137             variable_error (variable, _("bad absolute minimum"));
00138             goto exit_on_error;
00139         }
00140         if (variable->rangemin < variable->rangeminabs)
00141         {
00142             variable_error (variable, _("minimum range not allowed"));
00143             goto exit_on_error;
00144         }
00145     }
00146     else
00147     {
00148         variable_error (variable, _("no minimum range"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00152     {
00153         variable->rangemax
00154             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00155                                   &error_code);
00156         if (error_code)
00157         {
00158             variable_error (variable, _("bad maximum"));
00159             goto exit_on_error;
00160         }
00161         variable->rangemaxabs = xml_node_get_float_with_default
00162             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00163             &error_code);
00164         if (error_code)
00165         {
00166             variable_error (variable, _("bad absolute maximum"));
00167             goto exit_on_error;
00168         }
00169         if (variable->rangemax > variable->rangemaxabs)
00170         {
00171             variable_error (variable, _("maximum range not allowed"));
00172             goto exit_on_error;
00173         }
00174         if (variable->rangemax < variable->rangemin)
00175         {
00176             variable_error (variable, _("bad range"));
00177             goto exit_on_error;
00178         }
00179     }
00180     else
00181     {
00182         variable_error (variable, _("no maximum range"));
00183         goto exit_on_error;
00184     }
00185     variable->precision
00186         = xml_node_get_uint_with_default (node, (const xmlChar *)
00187                                           LABEL_PRECISION,
00188                                           DEFAULT_PRECISION, &error_code);
00189     if (error_code || variable->precision >= NPRECISIONS)
00190     {
00191         variable_error (variable, _("bad precision"));
00192         goto exit_on_error;
00193     }

```

```

00193     if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
00194     {
00195         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00196         {
00197             variable->nsweeps
00198             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
&error_code);
00199             if (error_code || !variable->nsweeps)
00200             {
00201                 variable_error (variable, _("bad sweeps"));
00202                 goto exit_on_error;
00203             }
00204         }
00205     }
00206     else
00207     {
00208         variable_error (variable, _("no sweeps number"));
00209         goto exit_on_error;
00210     }
00211     #if DEBUG_VARIABLE
00212     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00213     #endif
00214 }
00215 if (algorithm == ALGORITHM_GENETIC)
00216 {
00217     // Obtaining bits representing each variable
00218     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00219     {
00220         variable->nbits
00221         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
&error_code);
00222         if (error_code || !variable->nbits)
00223         {
00224             variable_error (variable, _("invalid bits number"));
00225             goto exit_on_error;
00226         }
00227     }
00228     else
00229     {
00230         variable_error (variable, _("no bits number"));
00231         goto exit_on_error;
00232     }
00233 }
00234 }
00235 else if (nsteps)
00236 {
00237     variable->step
00238     = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
00239     if (error_code || variable->step < 0.)
00240     {
00241         variable_error (variable, _("bad step size"));
00242         goto exit_on_error;
00243     }
00244 }
00245 }
00246 #if DEBUG_VARIABLE
00247 fprintf (stderr, "variable_open_xml: end\n");
00248 #endif
00249 return 1;
00250 exit_on_error:
00251 variable_free (variable, INPUT_TYPE_XML);
00252 #if DEBUG_VARIABLE
00253 fprintf (stderr, "variable_open_xml: end\n");
00254 #endif
00255 return 0;
00256 }
00257
00263 int
00264 variable_open_json (Variable * variable,
00265                     JsonNode * node,
00266                     unsigned int algorithm,
00267                     unsigned int nsteps)
00268 {
00269     JsonObject *object;
00270     const char *label;
00271     int error_code;
00272     #if DEBUG_VARIABLE
00273     fprintf (stderr, "variable_open_json: start\n");
00274     #endif
00275     object = json_node_get_object (node);
00276     label = json_object_get_string_member (object, LABEL_NAME);
00277     if (!label)
00278     {
00279         variable_error (variable, _("no name"));
00280         goto exit_on_error;
00281     }
00282     variable->name = g_strdup (label);

```

```

00284     if (json_object_get_member (object, LABEL_MINIMUM))
00285     {
00286         variable->rangemin
00287         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00288         if (error_code)
00289         {
00290             variable_error (variable, _("bad minimum"));
00291             goto exit_on_error;
00292         }
00293         variable->rangeminabs
00294         = json_object_get_float_with_default (object,
00295 LABEL_ABSOLUTE_MINIMUM,
00296                                             -G_MAXDOUBLE, &error_code);
00297         if (error_code)
00298         {
00299             variable_error (variable, _("bad absolute minimum"));
00300             goto exit_on_error;
00301         }
00302         if (variable->rangemin < variable->rangeminabs)
00303         {
00304             variable_error (variable, _("minimum range not allowed"));
00305             goto exit_on_error;
00306         }
00307     }
00308     else
00309     {
00310         variable_error (variable, _("no minimum range"));
00311         goto exit_on_error;
00312     }
00313     if (json_object_get_member (object, LABEL_MAXIMUM))
00314     {
00315         variable->rangemax
00316         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00317         if (error_code)
00318         {
00319             variable_error (variable, _("bad maximum"));
00320             goto exit_on_error;
00321         }
00322         variable->rangemaxabs
00323         = json_object_get_float_with_default (object,
00324 LABEL_ABSOLUTE_MAXIMUM,
00325                                             G_MAXDOUBLE, &error_code);
00326         if (error_code)
00327         {
00328             variable_error (variable, _("bad absolute maximum"));
00329             goto exit_on_error;
00330         }
00331         if (variable->rangemax > variable->rangemaxabs)
00332         {
00333             variable_error (variable, _("maximum range not allowed"));
00334             goto exit_on_error;
00335         }
00336         if (variable->rangemax < variable->rangemin)
00337         {
00338             variable_error (variable, _("bad range"));
00339             goto exit_on_error;
00340         }
00341     }
00342     else
00343     {
00344         variable_error (variable, _("no maximum range"));
00345         goto exit_on_error;
00346     }
00347     variable->precision
00348     = json_object_get_uint_with_default (object,
00349 LABEL_PRECISION,
00350                                         DEFAULT_PRECISION, &error_code);
00351     if (error_code || variable->precision >= NPRECISIONS)
00352     {
00353         variable_error (variable, _("bad precision"));
00354         goto exit_on_error;
00355     }
00356     if (algorithm == ALGORITHM_SWEEP || algorithm ==
00357 ALGORITHM_ORTHOGONAL)
00358     {
00359         if (json_object_get_member (object, LABEL_NSWEEPS))
00360         {
00361             variable->nsweeps
00362             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00363             if (error_code || !variable->nsweeps)
00364             {
00365                 variable_error (variable, _("bad sweeps"));
00366                 goto exit_on_error;
00367             }
00368         }
00369     }
00370     else
00371     {
00372         variable->nsweeps = 0;
00373     }

```

```

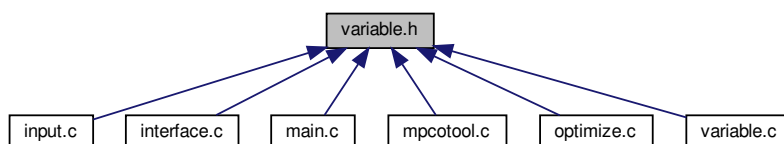
00367         variable_error (variable, _("no sweeps number"));
00368         goto exit_on_error;
00369     }
00370 #if DEBUG_VARIABLE
00371     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00372 #endif
00373 }
00374 if (algorithm == ALGORITHM_GENETIC)
00375 {
00376     // Obtaining bits representing each variable
00377     if (json_object_get_member (object, LABEL_NBITS))
00378     {
00379         variable->nbits
00380         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00381         if (error_code || !variable->nbits)
00382         {
00383             variable_error (variable, _("invalid bits number"));
00384             goto exit_on_error;
00385         }
00386     }
00387     else
00388     {
00389         variable_error (variable, _("no bits number"));
00390         goto exit_on_error;
00391     }
00392 }
00393 else if (nsteps)
00394 {
00395     variable->step = json_object_get_float (object,
00396     LABEL_STEP, &error_code);
00397     if (error_code || variable->step < 0.)
00398     {
00399         variable_error (variable, _("bad step size"));
00400         goto exit_on_error;
00401     }
00402 }
00403 #if DEBUG_VARIABLE
00404     fprintf (stderr, "variable_open_json: end\n");
00405 #endif
00406     return 1;
00407 exit_on_error:
00408     variable_free (variable, INPUT_TYPE_JSON);
00409 #if DEBUG_VARIABLE
00410     fprintf (stderr, "variable_open_json: end\n");
00411 #endif
00412     return 0;
00413 }

```

4.31 variable.h File Reference

Header file to define the variable data.

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



Data Structures

- struct [Variable](#)

Struct to define the variable data.

Enumerations

- enum [Algorithm](#) { [ALGORITHM_MONTE_CARLO](#) = 0, [ALGORITHM_SWEEP](#) = 1, [ALGORITHM_GENETIC](#) = 2, [ALGORITHM_ORTHOGONAL](#) = 3 }

Enum to define the algorithms.

Functions

- void [variable_free](#) ([Variable](#) *variable, unsigned int type)
- void [variable_error](#) ([Variable](#) *variable, char *message)
- int [variable_open_xml](#) ([Variable](#) *variable, xmlDoc *node, unsigned int algorithm, unsigned int nsteps)
- int [variable_open_json](#) ([Variable](#) *variable, cJSON *node, unsigned int algorithm, unsigned int nsteps)

Variables

- const char * [format](#) [[NPRECISIONS](#)]
Array of C-strings with variable formats.
- const double [precision](#) [[NPRECISIONS](#)]
Array of variable precisions.

4.31.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

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

4.31.2 Enumeration Type Documentation

4.31.2.1 Algorithm

enum [Algorithm](#)

Enum to define the algorithms.

Enumerator

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

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

```
00043 {
00044     ALGORITHM_MONTE_CARLO = 0,
00045     ALGORITHM_SWEEP = 1,
00046     ALGORITHM_GENETIC = 2,
00047     ALGORITHM_ORTHOGONAL = 3
00048 };
```

4.31.3 Function Documentation

4.31.3.1 variable_error()

```
void variable_error (
    Variable * variable,
    char * message )
```

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

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	Error message.

Definition at line 85 of file [variable.c](#).

```
00089 {
00090     char buffer[64];
00091     if (!variable->name)
00092         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00093     else
00094         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00095     error_message = g_strdup (buffer);
00096 }
```

4.31.3.2 variable_free()

```
void variable_free (
    Variable * variable,
    unsigned int type )
```

Function to free the memory of a [Variable](#) struct.

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	Type of input file.

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

```

00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_free: start\n");
00071     #endif
00072     if (type == INPUT_TYPE_XML)
00073         xmlFree (variable->name);
00074     else
00075         g_free (variable->name);
00076     #if DEBUG_VARIABLE
00077         fprintf (stderr, "variable_free: end\n");
00078     #endif
00079 }
```

4.31.3.3 variable_open_json()

```

int variable_open_json (
    Variable * variable,
    JsonNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

<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 264 of file [variable.c](#).

```

00269 {
00270     JsonObject *object;
00271     const char *label;
00272     int error_code;
00273     #if DEBUG_VARIABLE
00274         fprintf (stderr, "variable_open_json: start\n");
00275     #endif
00276     object = json_node_get_object (node);
00277     label = json_object_get_string_member (object, LABEL_NAME);
00278     if (!label)
00279     {
00280         variable_error (variable, _("no name"));
00281         goto exit_on_error;
00282     }
```

```

00282     }
00283     variable->name = g_strdup (label);
00284     if (json_object_get_member (object, LABEL_MINIMUM))
00285     {
00286         variable->rangemin
00287         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00288         if (error_code)
00289         {
00290             variable_error (variable, _("bad minimum"));
00291             goto exit_on_error;
00292         }
00293         variable->rangeminabs
00294         = json_object_get_float_with_default (object,
00295         LABEL_ABSOLUTE_MINIMUM,
00296         -G_MAXDOUBLE, &error_code);
00297         if (error_code)
00298         {
00299             variable_error (variable, _("bad absolute minimum"));
00300             goto exit_on_error;
00301         }
00302         if (variable->rangemin < variable->rangeminabs)
00303         {
00304             variable_error (variable, _("minimum range not allowed"));
00305             goto exit_on_error;
00306         }
00307     }
00308     else
00309     {
00310         variable_error (variable, _("no minimum range"));
00311         goto exit_on_error;
00312     }
00313     if (json_object_get_member (object, LABEL_MAXIMUM))
00314     {
00315         variable->rangemax
00316         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00317         if (error_code)
00318         {
00319             variable_error (variable, _("bad maximum"));
00320             goto exit_on_error;
00321         }
00322         variable->rangemaxabs
00323         = json_object_get_float_with_default (object,
00324         LABEL_ABSOLUTE_MAXIMUM,
00325         G_MAXDOUBLE, &error_code);
00326         if (error_code)
00327         {
00328             variable_error (variable, _("bad absolute maximum"));
00329             goto exit_on_error;
00330         }
00331         if (variable->rangemax > variable->rangemaxabs)
00332         {
00333             variable_error (variable, _("maximum range not allowed"));
00334             goto exit_on_error;
00335         }
00336         if (variable->rangemax < variable->rangemin)
00337         {
00338             variable_error (variable, _("bad range"));
00339             goto exit_on_error;
00340         }
00341     }
00342     else
00343     {
00344         variable_error (variable, _("no maximum range"));
00345         goto exit_on_error;
00346     }
00347     variable->precision
00348     = json_object_get_uint_with_default (object,
00349     LABEL_PRECISION,
00350     DEFAULT_PRECISION, &error_code);
00351     if (error_code || variable->precision >= NPRECISIONS)
00352     {
00353         variable_error (variable, _("bad precision"));
00354         goto exit_on_error;
00355     }
00356     if (algorithm == ALGORITHM_SWEEP || algorithm ==
00357     ALGORITHM_ORTHOGONAL)
00358     {
00359         if (json_object_get_member (object, LABEL_NSWEEPS))
00360         {
00361             variable->nsweeps
00362             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00363             if (error_code || !variable->nsweeps)
00364             {
00365                 variable_error (variable, _("bad sweeps"));
00366                 goto exit_on_error;
00367             }
00368         }
00369     }

```

```

00365     else
00366     {
00367         variable_error (variable, _("no sweeps number"));
00368         goto exit_on_error;
00369     }
00370 #if DEBUG_VARIABLE
00371     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00372 #endif
00373 }
00374 if (algorithm == ALGORITHM_GENETIC)
00375 {
00376     // Obtaining bits representing each variable
00377     if (json_object_get_member (object, LABEL_NBITS))
00378     {
00379         variable->nbits
00380         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00381         if (error_code || !variable->nbits)
00382         {
00383             variable_error (variable, _("invalid bits number"));
00384             goto exit_on_error;
00385         }
00386     }
00387     else
00388     {
00389         variable_error (variable, _("no bits number"));
00390         goto exit_on_error;
00391     }
00392 }
00393 else if (nsteps)
00394 {
00395     variable->step = json_object_get_float (object,
00396 LABEL_STEP, &error_code);
00397     if (error_code || variable->step < 0.)
00398     {
00399         variable_error (variable, _("bad step size"));
00400         goto exit_on_error;
00401     }
00402 }
00403 #if DEBUG_VARIABLE
00404     fprintf (stderr, "variable_open_json: end\n");
00405 #endif
00406     return 1;
00407 exit_on_error:
00408     variable_free (variable, INPUT_TYPE_JSON);
00409 #if DEBUG_VARIABLE
00410     fprintf (stderr, "variable_open_json: end\n");
00411 #endif
00412     return 0;
00413 }

```

Here is the call graph for this function:



4.31.3.4 variable_open_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

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

Definition at line 104 of file [variable.c](#).

```

00109 {
00110     int error_code;
00111
00112     #if DEBUG_VARIABLE
00113         fprintf (stderr, "variable_open_xml: start\n");
00114     #endif
00115
00116     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00117     if (!variable->name)
00118     {
00119         variable_error (variable, _("no name"));
00120         goto exit_on_error;
00121     }
00122     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00123     {
00124         variable->rangemin
00125             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00126                                   &error_code);
00127         if (error_code)
00128         {
00129             variable_error (variable, _("bad minimum"));
00130             goto exit_on_error;
00131         }
00132         variable->rangeminabs = xml_node_get_float_with_default
00133             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00134             &error_code);
00135         if (error_code)
00136         {
00137             variable_error (variable, _("bad absolute minimum"));
00138             goto exit_on_error;
00139         }
00140         if (variable->rangemin < variable->rangeminabs)
00141         {
00142             variable_error (variable, _("minimum range not allowed"));
00143             goto exit_on_error;
00144         }
00145     }
00146     else
00147     {
00148         variable_error (variable, _("no minimum range"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00152     {
00153         variable->rangemax
00154             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00155                                   &error_code);
00156         if (error_code)
00157         {
00158             variable_error (variable, _("bad maximum"));
00159             goto exit_on_error;
00160         }
00161         variable->rangemaxabs = xml_node_get_float_with_default
00162             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00163             &error_code);
00164         if (error_code)
00165         {
00166             variable_error (variable, _("bad absolute maximum"));
00167             goto exit_on_error;
00168         }
00169         if (variable->rangemax > variable->rangemaxabs)
00170         {
00171             variable_error (variable, _("maximum range not allowed"));
00172             goto exit_on_error;

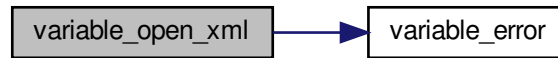
```

```

00173     }
00174     if (variable->rangemax < variable->rangemin)
00175     {
00176         variable_error (variable, _("bad range"));
00177         goto exit_on_error;
00178     }
00179 }
00180 else
00181 {
00182     variable_error (variable, _("no maximum range"));
00183     goto exit_on_error;
00184 }
00185 variable->precision
00186 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                DEFAULT_PRECISION, &error_code);
00187 if (error_code || variable->precision >= NPRECISIONS)
00188 {
00189     variable_error (variable, _("bad precision"));
00190     goto exit_on_error;
00191 }
00192 }
00193 if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
00194 {
00195     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00196     {
00197         variable->nsweeps
00198         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
                                &error_code);
00199         if (error_code || !variable->nsweeps)
00200         {
00201             variable_error (variable, _("bad sweeps"));
00202             goto exit_on_error;
00203         }
00204     }
00205 }
00206 else
00207 {
00208     variable_error (variable, _("no sweeps number"));
00209     goto exit_on_error;
00210 }
00211 #if DEBUG_VARIABLE
00212 fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00213 #endif
00214 }
00215 if (algorithm == ALGORITHM_GENETIC)
00216 {
00217     // Obtaining bits representing each variable
00218     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00219     {
00220         variable->nbits
00221         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                &error_code);
00222         if (error_code || !variable->nbits)
00223         {
00224             variable_error (variable, _("invalid bits number"));
00225             goto exit_on_error;
00226         }
00227     }
00228 }
00229 else
00230 {
00231     variable_error (variable, _("no bits number"));
00232     goto exit_on_error;
00233 }
00234 }
00235 else if (nsteps)
00236 {
00237     variable->step
00238     = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
00239     if (error_code || variable->step < 0.)
00240     {
00241         variable_error (variable, _("bad step size"));
00242         goto exit_on_error;
00243     }
00244 }
00245 }
00246 #if DEBUG_VARIABLE
00247 fprintf (stderr, "variable_open_xml: end\n");
00248 #endif
00249 return 1;
00250 exit_on_error:
00251 variable_free (variable, INPUT_TYPE_XML);
00252 #if DEBUG_VARIABLE
00253 fprintf (stderr, "variable_open_xml: end\n");
00254 #endif
00255 return 0;
00256 }

```

Here is the call graph for this function:



4.32 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 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 VARIABLE__H
00033 #define VARIABLE__H 1
00034
00035 enum Algorithm
00036 {
00037     ALGORITHM_MONTE_CARLO = 0,
00038     ALGORITHM_SWEEP = 1,
00039     ALGORITHM_GENETIC = 2,
00040     ALGORITHM_ORTHOGONAL = 3
00041 };
00042
00043 typedef struct
00044 {
00045     char *name;
00046     double rangemin;
00047     double rangemax;
00048     double rangeminabs;
00049     double rangemaxabs;
00050     double step;
00051     unsigned int precision;
00052     unsigned int nsweeps;
00053     unsigned int nbits;
00054 } Variable;
00055
00056 extern const char *format[NPRECISIONS];
00057 extern const double precision[NPRECISIONS];
00058
00059 // Public functions
00060 void variable_free (Variable * variable, unsigned int type);
00061 void variable_error (Variable * variable, char *message);
00062 int variable_open_xml (Variable * variable, xmlNode * node,
  
```

```
00074             unsigned int algorithm, unsigned int nsteps);
00075 int variable_open_json (Variable * variable, JsonNode * node,
00076             unsigned int algorithm, unsigned int nsteps);
00077
00078 #endif
```


Index

Algorithm
 variable.h, 286

ClimbingMethod
 input.h, 70

config.h, 19
 INPUT_TYPE, 22

cores_number
 utils.c, 239
 utils.h, 259

ErrorNorm
 input.h, 70

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_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_open_json
 experiment.c, 27
 experiment.h, 37

experiment_open_xml
 experiment.c, 29
 experiment.h, 39

format
 variable.c, 280

gtk_array_get_active
 utils.c, 240
 utils.h, 259

INPUT_TYPE
 config.h, 22

Input, 6

input.c, 42
 input_error, 43
 input_free, 43
 input_new, 44
 input_open, 45
 input_open_json, 46
 input_open_xml, 51

input.h, 69
 ClimbingMethod, 70
 ErrorNorm, 70
 input_free, 71
 input_new, 72
 input_open, 72

input_error
 input.c, 43

input_free
 input.c, 43
 input.h, 71

input_new
 input.c, 44
 input.h, 72

input_open
 input.c, 45
 input.h, 72

input_open_json
 input.c, 46

input_open_xml
 input.c, 51

input_save
 interface.c, 77

input_save_climbing_json
 interface.c, 78

input_save_climbing_xml
 interface.c, 79

input_save_json
 interface.c, 80

input_save_xml
 interface.c, 82

interface.c, 75
 input_save, 77
 input_save_climbing_json, 78
 input_save_climbing_xml, 79
 input_save_json, 80
 input_save_xml, 82
 options_new, 85
 running_new, 86
 window_about, 87
 window_add_experiment, 87
 window_add_variable, 89

- window_get_algorithm, 89
 - window_get_climbing, 90
 - window_get_norm, 91
 - window_help, 91
 - window_inputs_experiment, 92
 - window_label_variable, 92
 - window_name_experiment, 93
 - window_new, 93
 - window_open, 102
 - window_precision_variable, 103
 - window_rangemax_variable, 104
 - window_rangemaxabs_variable, 104
 - window_rangemin_variable, 104
 - window_rangeminabs_variable, 105
 - window_read, 105
 - window_remove_experiment, 107
 - window_remove_variable, 108
 - window_run, 109
 - window_save, 110
 - window_save_climbing, 112
 - window_set_algorithm, 113
 - window_set_experiment, 114
 - window_set_variable, 114
 - window_step_variable, 116
 - window_template_experiment, 116
 - window_update, 117
 - window_update_climbing, 119
 - window_update_variable, 120
 - window_weight_experiment, 120
- interface.h, 153
 - window_new, 154
- json_object_get_float
 - utils.c, 240
 - utils.h, 260
- json_object_get_float_with_default
 - utils.c, 241
 - utils.h, 260
- json_object_get_int
 - utils.c, 242
 - utils.h, 261
- json_object_get_uint
 - utils.c, 243
 - utils.h, 262
- json_object_get_uint_with_default
 - utils.c, 243
 - utils.h, 262
- json_object_set_float
 - utils.c, 244
 - utils.h, 263
- json_object_set_int
 - utils.c, 245
 - utils.h, 264
- json_object_set_uint
 - utils.c, 245
 - utils.h, 264
- main.c, 165
- mpcotool
 - mpcotool.c, 168
 - mpcotool.h, 175
- mpcotool.c, 167
 - mpcotool, 168
- mpcotool.h, 174
 - mpcotool, 175
- Optimize, 7
 - thread_climbing, 10
- optimize.c, 178
 - optimize_MonteCarlo, 192
 - optimize_best, 180
 - optimize_best_climbing, 181
 - optimize_climbing, 181
 - optimize_climbing_sequential, 182
 - optimize_climbing_thread, 183
 - optimize_estimate_climbing_coordinates, 184
 - optimize_estimate_climbing_random, 185
 - optimize_free, 185
 - optimize_genetic, 186
 - optimize_genetic_objective, 187
 - optimize_input, 188
 - optimize_iterate, 189
 - optimize_merge, 190
 - optimize_merge_old, 191
 - optimize_norm_euclidian, 193
 - optimize_norm_maximum, 194
 - optimize_norm_p, 194
 - optimize_norm_taxicab, 195
 - optimize_open, 196
 - optimize_orthogonal, 200
 - optimize_parse, 201
 - optimize_print, 203
 - optimize_refine, 204
 - optimize_save_old, 205
 - optimize_save_variables, 206
 - optimize_sequential, 206
 - optimize_step, 207
 - optimize_step_climbing, 207
 - optimize_sweep, 209
 - optimize_synchronise, 210
 - optimize_thread, 211
- optimize.h, 230
 - optimize_free, 232
 - optimize_open, 232
- optimize_MonteCarlo
 - optimize.c, 192
- optimize_best
 - optimize.c, 180
- optimize_best_climbing
 - optimize.c, 181
- optimize_climbing
 - optimize.c, 181
- optimize_climbing_sequential
 - optimize.c, 182
- optimize_climbing_thread
 - optimize.c, 183
- optimize_estimate_climbing_coordinates
 - optimize.c, 184

- optimize_estimate_climbing_random
 - optimize.c, 185
- optimize_free
 - optimize.c, 185
 - optimize.h, 232
- optimize_genetic
 - optimize.c, 186
- optimize_genetic_objective
 - optimize.c, 187
- optimize_input
 - optimize.c, 188
- optimize_iterate
 - optimize.c, 189
- optimize_merge
 - optimize.c, 190
- optimize_merge_old
 - optimize.c, 191
- optimize_norm_euclidian
 - optimize.c, 193
- optimize_norm_maximum
 - optimize.c, 194
- optimize_norm_p
 - optimize.c, 194
- optimize_norm_taxicab
 - optimize.c, 195
- optimize_open
 - optimize.c, 196
 - optimize.h, 232
- optimize_orthogonal
 - optimize.c, 200
- optimize_parse
 - optimize.c, 201
- optimize_print
 - optimize.c, 203
- optimize_refine
 - optimize.c, 204
- optimize_save_old
 - optimize.c, 205
- optimize_save_variables
 - optimize.c, 206
- optimize_sequential
 - optimize.c, 206
- optimize_step
 - optimize.c, 207
- optimize_step_climbing
 - optimize.c, 207
- optimize_sweep
 - optimize.c, 209
- optimize_synchronise
 - optimize.c, 210
- optimize_thread
 - optimize.c, 211
- Options, 11
- options_new
 - interface.c, 85
- ParallelData, 11
- precision
 - variable.c, 280
- process_pending
 - utils.c, 246
 - utils.h, 265
- Running, 12
- running_new
 - interface.c, 86
- show_error
 - utils.c, 246
 - utils.h, 265
- show_message
 - utils.c, 246
 - utils.h, 266
- stencil
 - experiment.c, 31
- thread_climbing
 - Optimize, 10
- utils.c, 238
 - cores_number, 239
 - gtk_array_get_active, 240
 - json_object_get_float, 240
 - json_object_get_float_with_default, 241
 - json_object_get_int, 242
 - json_object_get_uint, 243
 - json_object_get_uint_with_default, 243
 - json_object_set_float, 244
 - json_object_set_int, 245
 - json_object_set_uint, 245
 - process_pending, 246
 - show_error, 246
 - show_message, 246
 - xml_node_get_float, 247
 - xml_node_get_float_with_default, 248
 - xml_node_get_int, 249
 - xml_node_get_uint, 249
 - xml_node_get_uint_with_default, 250
 - xml_node_set_float, 251
 - xml_node_set_int, 252
 - xml_node_set_uint, 252
- utils.h, 257
 - cores_number, 259
 - gtk_array_get_active, 259
 - json_object_get_float, 260
 - json_object_get_float_with_default, 260
 - json_object_get_int, 261
 - json_object_get_uint, 262
 - json_object_get_uint_with_default, 262
 - json_object_set_float, 263
 - json_object_set_int, 264
 - json_object_set_uint, 264
 - process_pending, 265
 - show_error, 265
 - show_message, 266
 - xml_node_get_float, 266
 - xml_node_get_float_with_default, 267
 - xml_node_get_int, 268

- xml_node_get_uint, [269](#)
- xml_node_get_uint_with_default, [269](#)
- xml_node_set_float, [270](#)
- xml_node_set_int, [271](#)
- xml_node_set_uint, [271](#)
- Variable, [12](#)
- variable.c, [273](#)
 - format, [280](#)
 - precision, [280](#)
 - variable_error, [274](#)
 - variable_free, [274](#)
 - variable_open_json, [275](#)
 - variable_open_xml, [277](#)
- variable.h, [285](#)
 - Algorithm, [286](#)
 - variable_error, [287](#)
 - variable_free, [287](#)
 - variable_open_json, [288](#)
 - variable_open_xml, [290](#)
- variable_error
 - variable.c, [274](#)
 - variable.h, [287](#)
- variable_free
 - variable.c, [274](#)
 - variable.h, [287](#)
- variable_open_json
 - variable.c, [275](#)
 - variable.h, [288](#)
- variable_open_xml
 - variable.c, [277](#)
 - variable.h, [290](#)
- Window, [13](#)
- window_about
 - interface.c, [87](#)
- window_add_experiment
 - interface.c, [87](#)
- window_add_variable
 - interface.c, [89](#)
- window_get_algorithm
 - interface.c, [89](#)
- window_get_climbing
 - interface.c, [90](#)
- window_get_norm
 - interface.c, [91](#)
- window_help
 - interface.c, [91](#)
- window_inputs_experiment
 - interface.c, [92](#)
- window_label_variable
 - interface.c, [92](#)
- window_name_experiment
 - interface.c, [93](#)
- window_new
 - interface.c, [93](#)
 - interface.h, [154](#)
- window_open
 - interface.c, [102](#)
- window_precision_variable
 - interface.c, [103](#)
- window_rangemax_variable
 - interface.c, [104](#)
- window_rangemaxabs_variable
 - interface.c, [104](#)
- window_rangemin_variable
 - interface.c, [104](#)
- window_rangeminabs_variable
 - interface.c, [105](#)
- window_read
 - interface.c, [105](#)
- window_remove_experiment
 - interface.c, [107](#)
- window_remove_variable
 - interface.c, [108](#)
- window_run
 - interface.c, [109](#)
- window_save
 - interface.c, [110](#)
- window_save_climbing
 - interface.c, [112](#)
- window_set_algorithm
 - interface.c, [113](#)
- window_set_experiment
 - interface.c, [114](#)
- window_set_variable
 - interface.c, [114](#)
- window_step_variable
 - interface.c, [116](#)
- window_template_experiment
 - interface.c, [116](#)
- window_update
 - interface.c, [117](#)
- window_update_climbing
 - interface.c, [119](#)
- window_update_variable
 - interface.c, [120](#)
- window_weight_experiment
 - interface.c, [120](#)
- xml_node_get_float
 - utils.c, [247](#)
 - utils.h, [266](#)
- xml_node_get_float_with_default
 - utils.c, [248](#)
 - utils.h, [267](#)
- xml_node_get_int
 - utils.c, [249](#)
 - utils.h, [268](#)
- xml_node_get_uint
 - utils.c, [249](#)
 - utils.h, [269](#)
- xml_node_get_uint_with_default
 - utils.c, [250](#)
 - utils.h, [269](#)
- xml_node_set_float
 - utils.c, [251](#)
 - utils.h, [270](#)

xml_node_set_int
 utils.c, [252](#)
 utils.h, [271](#)
xml_node_set_uint
 utils.c, [252](#)
 utils.h, [271](#)