

MPCOTool

3.4.0

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_direction	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()	26
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	template	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()	37
4.5.2.3	experiment_new()	37
4.5.2.4	experiment_open_json()	38
4.5.2.5	experiment_open_xml()	40
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_open()	44
4.7.2.3	input_open_json()	45
4.7.2.4	input_open_xml()	50
4.8	input.c	56
4.9	input.h File Reference	67
4.9.1	Detailed Description	69
4.9.2	Enumeration Type Documentation	69
4.9.2.1	DirectionMethod	69
4.9.2.2	ErrorNorm	69
4.9.3	Function Documentation	70
4.9.3.1	input_error()	70
4.9.3.2	input_open()	70
4.9.3.3	input_open_json()	71
4.9.3.4	input_open_xml()	77
4.10	input.h	82
4.11	interface.c File Reference	84
4.11.1	Detailed Description	86
4.11.2	Function Documentation	86
4.11.2.1	input_save()	86
4.11.2.2	input_save_direction_json()	88
4.11.2.3	input_save_direction_xml()	88
4.11.2.4	input_save_json()	89
4.11.2.5	input_save_xml()	92
4.11.2.6	window_get_algorithm()	95
4.11.2.7	window_get_direction()	95
4.11.2.8	window_get_norm()	96
4.11.2.9	window_new()	97
4.11.2.10	window_read()	106
4.11.2.11	window_save()	108
4.11.2.12	window_template_experiment()	110

4.12 interface.c	110
4.13 interface.h File Reference	142
4.13.1 Detailed Description	145
4.13.2 Function Documentation	145
4.13.2.1 gtk_array_get_active()	145
4.13.2.2 input_save()	145
4.13.2.3 window_get_algorithm()	147
4.13.2.4 window_get_direction()	147
4.13.2.5 window_get_norm()	148
4.13.2.6 window_new()	149
4.13.2.7 window_read()	158
4.13.2.8 window_save()	160
4.13.2.9 window_template_experiment()	162
4.14 interface.h	162
4.15 main.c File Reference	165
4.15.1 Detailed Description	166
4.16 main.c	166
4.17 optimize.c File Reference	169
4.17.1 Detailed Description	172
4.17.2 Function Documentation	172
4.17.2.1 optimize_best()	172
4.17.2.2 optimize_best_direction()	173
4.17.2.3 optimize_direction_sequential()	173
4.17.2.4 optimize_direction_thread()	174
4.17.2.5 optimize_estimate_direction_coordinates()	176
4.17.2.6 optimize_estimate_direction_random()	177
4.17.2.7 optimize_genetic_objective()	178
4.17.2.8 optimize_input()	179
4.17.2.9 optimize_merge()	181
4.17.2.10 optimize_norm_euclidian()	182

4.17.2.11	<code>optimize_norm_maximum()</code>	182
4.17.2.12	<code>optimize_norm_p()</code>	183
4.17.2.13	<code>optimize_norm_taxicab()</code>	184
4.17.2.14	<code>optimize_parse()</code>	185
4.17.2.15	<code>optimize_save_variables()</code>	187
4.17.2.16	<code>optimize_step_direction()</code>	187
4.17.2.17	<code>optimize_thread()</code>	189
4.18	<code>optimize.c</code>	190
4.19	<code>optimize.h</code> File Reference	208
4.19.1	Detailed Description	210
4.19.2	Function Documentation	210
4.19.2.1	<code>optimize_best()</code>	210
4.19.2.2	<code>optimize_best_direction()</code>	211
4.19.2.3	<code>optimize_direction_sequential()</code>	212
4.19.2.4	<code>optimize_direction_thread()</code>	213
4.19.2.5	<code>optimize_estimate_direction_coordinates()</code>	214
4.19.2.6	<code>optimize_estimate_direction_random()</code>	215
4.19.2.7	<code>optimize_genetic_objective()</code>	215
4.19.2.8	<code>optimize_input()</code>	216
4.19.2.9	<code>optimize_merge()</code>	217
4.19.2.10	<code>optimize_norm_euclidian()</code>	219
4.19.2.11	<code>optimize_norm_maximum()</code>	220
4.19.2.12	<code>optimize_norm_p()</code>	221
4.19.2.13	<code>optimize_norm_taxicab()</code>	222
4.19.2.14	<code>optimize_parse()</code>	223
4.19.2.15	<code>optimize_save_variables()</code>	225
4.19.2.16	<code>optimize_step_direction()</code>	225
4.19.2.17	<code>optimize_thread()</code>	227
4.20	<code>optimize.h</code>	228
4.21	<code>utils.c</code> File Reference	229

4.21.1 Detailed Description	231
4.21.2 Function Documentation	231
4.21.2.1 cores_number()	231
4.21.2.2 gtk_array_get_active()	232
4.21.2.3 json_object_get_float()	232
4.21.2.4 json_object_get_float_with_default()	233
4.21.2.5 json_object_get_int()	234
4.21.2.6 json_object_get_uint()	235
4.21.2.7 json_object_get_uint_with_default()	235
4.21.2.8 json_object_set_float()	236
4.21.2.9 json_object_set_int()	237
4.21.2.10 json_object_set_uint()	237
4.21.2.11 show_error()	238
4.21.2.12 show_message()	238
4.21.2.13 xml_node_get_float()	239
4.21.2.14 xml_node_get_float_with_default()	240
4.21.2.15 xml_node_get_int()	240
4.21.2.16 xml_node_get_uint()	241
4.21.2.17 xml_node_get_uint_with_default()	242
4.21.2.18 xml_node_set_float()	243
4.21.2.19 xml_node_set_int()	243
4.21.2.20 xml_node_set_uint()	244
4.22 utils.c	244
4.23 utils.h File Reference	248
4.23.1 Detailed Description	250
4.23.2 Function Documentation	250
4.23.2.1 cores_number()	250
4.23.2.2 gtk_array_get_active()	250
4.23.2.3 json_object_get_float()	251
4.23.2.4 json_object_get_float_with_default()	252

4.23.2.5	json_object_get_int()	252
4.23.2.6	json_object_get_uint()	253
4.23.2.7	json_object_get_uint_with_default()	254
4.23.2.8	json_object_set_float()	255
4.23.2.9	json_object_set_int()	255
4.23.2.10	json_object_set_uint()	256
4.23.2.11	show_error()	256
4.23.2.12	show_message()	257
4.23.2.13	xml_node_get_float()	257
4.23.2.14	xml_node_get_float_with_default()	258
4.23.2.15	xml_node_get_int()	259
4.23.2.16	xml_node_get_uint()	260
4.23.2.17	xml_node_get_uint_with_default()	260
4.23.2.18	xml_node_set_float()	261
4.23.2.19	xml_node_set_int()	262
4.23.2.20	xml_node_set_uint()	262
4.24	utils.h	262
4.25	variable.c File Reference	264
4.25.1	Detailed Description	265
4.25.2	Function Documentation	265
4.25.2.1	variable_error()	265
4.25.2.2	variable_free()	265
4.25.2.3	variable_new()	266
4.25.2.4	variable_open_json()	266
4.25.2.5	variable_open_xml()	269
4.25.3	Variable Documentation	271
4.25.3.1	format	271
4.25.3.2	precision	272
4.26	variable.c	272
4.27	variable.h File Reference	277
4.27.1	Detailed Description	278
4.27.2	Enumeration Type Documentation	278
4.27.2.1	Algorithm	278
4.27.3	Function Documentation	278
4.27.3.1	variable_error()	278
4.27.3.2	variable_free()	279
4.27.3.3	variable_new()	279
4.27.3.4	variable_open_json()	280
4.27.3.5	variable_open_xml()	282
4.28	variable.h	285

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	67
interface.c	Source file to define the graphical interface functions	84
interface.h	Header file to define the graphical interface functions	142
main.c	Main source file	165
optimize.c	Source file to define the optimization functions	169
optimize.h	Header file to define the optimization functions	208
utils.c	Source file to define some useful functions	229
utils.h	Header file to define some useful functions	248
variable.c	Source file to define the variable data	264
variable.h	Header file to define the variable data	277

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 * [template](#) [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:

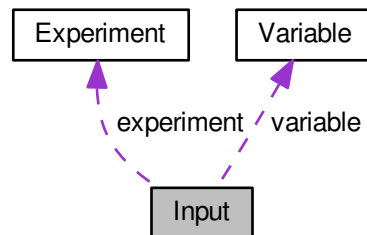
- [experiment.h](#)

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 direction search method.
- unsigned int [direction](#)
Method to estimate the direction search.
- unsigned int [nestimates](#)
Number of simulations to estimate the direction search.
- 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 [71](#) of file [input.h](#).

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

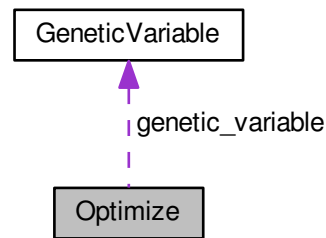
- [input.h](#)

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 direction search method step sizes.
- double * [direction](#)
Vector of direction search 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_direction](#)
- 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 direction search method.

- unsigned int [nestimates](#)
Number of simulations to estimate the direction.
- 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_direction](#)
Beginning simulation number of the task for the direction search method.
- unsigned int [nend_direction](#)
Ending simulation number of the task for the direction search 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_direction

```
unsigned int* Optimize::thread_direction
```

Array of simulation numbers to calculate on the thread for the direction search method.

Definition at line 80 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_direction`
Direction threads number GtkLabel.
- `GtkSpinButton * spin_direction`
Direction 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 122 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 67 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](#)
Direction search 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 56 of file [variable.h](#).

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

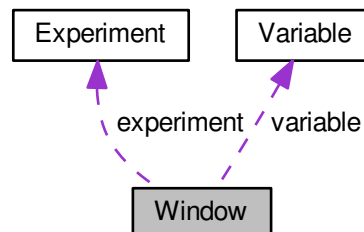
- [variable.h](#)

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_bests](#)
 - GtkLabel* to set the best number.
- GtkSpinButton * [spin_bests](#)
 - GtkSpinButton* to set the best number.
- GtkLabel * [label_population](#)
 - GtkLabel* to set the population number.
- GtkSpinButton * [spin_population](#)
 - GtkSpinButton* to set the population number.
- GtkLabel * [label_generations](#)
 - GtkLabel* to set the generations number.
- GtkSpinButton * [spin_generations](#)
 - GtkSpinButton* to set the generations number.
- GtkLabel * [label_mutation](#)
 - GtkLabel* to set the mutation ratio.
- GtkSpinButton * [spin_mutation](#)
 - GtkSpinButton* to set the mutation ratio.
- GtkLabel * [label_reproduction](#)
 - GtkLabel* to set the reproduction ratio.
- GtkSpinButton * [spin_reproduction](#)
 - GtkSpinButton* to set the reproduction ratio.
- GtkLabel * [label_adaptation](#)
 - GtkLabel* to set the adaptation ratio.
- GtkSpinButton * [spin_adaptation](#)
 - GtkSpinButton* to set the adaptation ratio.
- GtkCheckButton * [check_direction](#)
 - GtkCheckButton* to check running the direction search method.
- GtkGrid * [grid_direction](#)
 - GtkGrid* to pack the direction search method widgets.
- GtkRadioButton * [button_direction](#) [NDIRECTIONS]
 - GtkRadioButtons* array to set the direction estimate method.
- GtkLabel * [label_steps](#)
 - GtkLabel* to set the steps number.

- GtkSpinButton * [spin_steps](#)
GtkSpinButton to set the steps number.
- GtkLabel * [label_estimates](#)
GtkLabel to set the estimates number.
- GtkSpinButton * [spin_estimates](#)
GtkSpinButton to set the estimates number.
- GtkLabel * [label_relaxation](#)
GtkLabel to set the relaxation parameter.
- GtkSpinButton * [spin_relaxation](#)
GtkSpinButton to set the relaxation parameter.
- GtkLabel * [label_threshold](#)
GtkLabel to set the threshold.
- GtkSpinButton * [spin_threshold](#)
GtkSpinButton to set the threshold.
- GtkScrolledWindow * [scrolled_threshold](#)
GtkScrolledWindow to set the threshold.
- GtkFrame * [frame_variable](#)
Variable GtkFrame.
- GtkGrid * [grid_variable](#)
Variable GtkGrid.
- GtkComboBoxText * [combo_variable](#)
GtkComboBoxEntry to select a variable.
- GtkButton * [button_add_variable](#)
GtkButton to add a variable.
- GtkButton * [button_remove_variable](#)
GtkButton to remove a variable.
- GtkLabel * [label_variable](#)
Variable GtkLabel.
- GtkEntry * [entry_variable](#)
GtkEntry to set the variable name.
- GtkLabel * [label_min](#)
Minimum GtkLabel.
- GtkSpinButton * [spin_min](#)
Minimum GtkSpinButton.
- GtkScrolledWindow * [scrolled_min](#)
Minimum GtkScrolledWindow.
- GtkLabel * [label_max](#)
Maximum GtkLabel.
- GtkSpinButton * [spin_max](#)
Maximum GtkSpinButton.
- GtkScrolledWindow * [scrolled_max](#)
Maximum GtkScrolledWindow.
- GtkCheckButton * [check_minabs](#)
Absolute minimum GtkCheckButton.
- GtkSpinButton * [spin_minabs](#)
Absolute minimum GtkSpinButton.
- GtkScrolledWindow * [scrolled_minabs](#)
Absolute minimum GtkScrolledWindow.
- GtkCheckButton * [check_maxabs](#)
Absolute maximum GtkCheckButton.
- GtkSpinButton * [spin_maxabs](#)

- Absolute maximum GtkSpinButton.*
- GtkWidget * [scrolled_maxabs](#)
 - Absolute maximum GtkWidget.*
- GtkWidget * [label_precision](#)
 - Precision GtkWidget.*
- GtkWidget * [spin_precision](#)
 - Precision digits GtkWidget.*
- GtkWidget * [label_sweeps](#)
 - Sweeps number GtkWidget.*
- GtkWidget * [spin_sweeps](#)
 - Sweeps number GtkWidget.*
- GtkWidget * [label_bits](#)
 - Bits number GtkWidget.*
- GtkWidget * [spin_bits](#)
 - Bits number GtkWidget.*
- GtkWidget * [label_step](#)
 - GtkWidget to set the step.*
- GtkWidget * [spin_step](#)
 - GtkWidget to set the step.*
- GtkWidget * [scrolled_step](#)
 - step GtkWidget.*
- GtkWidget * [frame_experiment](#)
 - Experiment GtkWidget.*
- GtkWidget * [grid_experiment](#)
 - Experiment GtkWidget.*
- GtkWidget * [combo_experiment](#)
 - Experiment GtkWidgetEntry.*
- GtkWidget * [button_add_experiment](#)
 - GtkWidget to add a experiment.*
- GtkWidget * [button_remove_experiment](#)
 - GtkWidget to remove a experiment.*
- GtkWidget * [label_experiment](#)
 - Experiment GtkWidget.*
- GtkWidget * [button_experiment](#)
 - GtkFileChooserButton to set the experimental data file.*
- GtkWidget * [label_weight](#)
 - Weight GtkWidget.*
- GtkWidget * [spin_weight](#)
 - Weight GtkWidget.*
- GtkWidget * [check_template](#) [MAX_NINPUTS]
 - Array of GtkWidgetButtons to set the input templates.*
- GtkWidget * [button_template](#) [MAX_NINPUTS]
 - Array of GtkFileChooserButtons to set the input templates.*
- GdkPixbuf * [logo](#)
 - Logo GdkPixbuf.*
- [Experiment](#) * [experiment](#)
 - Array of experiments data.*
- [Variable](#) * [variable](#)
 - Array of variables data.*
- char * [application_directory](#)
 - Application directory.*

- gulong [id_experiment](#)
Identifier of the combo_experiment signal.
- gulong [id_experiment_name](#)
Identifier of the button_experiment signal.
- gulong [id_variable](#)
Identifier of the combo_variable signal.
- gulong [id_variable_label](#)
Identifier of the entry_variable signal.
- gulong [id_template](#) [MAX_NINPUTS]
Array of identifiers of the check_template signal.
- gulong [id_input](#) [MAX_NINPUTS]
Array of identifiers of the button_template signal.
- unsigned int [nexperiments](#)
Number of experiments.
- unsigned int [nvariables](#)
Number of variables.

3.8.1 Detailed Description

Struct to define the main window.

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

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

- [interface.h](#)

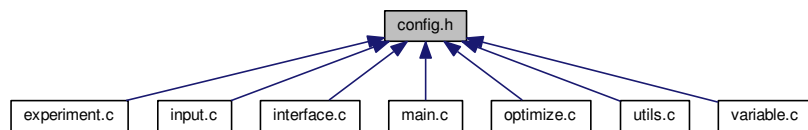
Chapter 4

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 3`
Number of stochastic algorithms.
- `#define NDIRECTIONS 2`
Number of direction 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_OPTIMIZE "optimize"
optimize label.
- #define LABEL_COORDINATES "coordinates"
coordinates label.
- #define LABEL_DIRECTION "direction"
direction 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_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 2012-2016, all rights reserved.

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

4.1.2 Enumeration Type Documentation

4.1.2.1 INPUT_TYPE

`enum INPUT_TYPE`

Enum to define the input file types.

Enumerator

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

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

```
00129 {
00130     INPUT_TYPE_XML = 0,
00131     INPUT_TYPE_JSON = 1
00132 };
```


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 2012-2016, 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 3
00043 #define NDIRECTIONS 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_OPTIMIZE "optimize"
00063 #define LABEL_COORDINATES "coordinates"
00064 #define LABEL_DIRECTION "direction"
00065 #define LABEL_EUCLIDIAN "euclidian"
00066 #define LABEL_EVALUATOR "evaluator"
00067 #define LABEL_EXPERIMENT "experiment"
00068 #define LABEL_EXPERIMENTS "experiments"
00069 #define LABEL_GENETIC "genetic"
00070 #define LABEL_MINIMUM "minimum"
00071 #define LABEL_MAXIMUM "maximum"
00072 #define LABEL_MONTE_CARLO "Monte-Carlo"
00073 #define LABEL_MUTATION "mutation"
00074 #define LABEL_NAME "name"
00075 #define LABEL_NBEST "nbest"
00076 #define LABEL_NBITS "nbits"
00077 #define LABEL_NESTIMATES "nestimates"
00078 #define LABEL_NGENERATIONS "ngenerations"
00079 #define LABEL_NITERATIONS "niterations"
00080 #define LABEL_NORM "norm"
00081 #define LABEL_NPOPULATION "npopulation"
00082 #define LABEL_NSIMULATIONS "nsimulations"

```

```

00094 #define LABEL_NSTEPS "nsteps"
00095 #define LABEL_NSWEEPS "nsweeps"
00096 #define LABEL_P "p"
00097 #define LABEL_PRECISION "precision"
00098 #define LABEL_RANDOM "random"
00099 #define LABEL_RELAXATION "relaxation"
00100 #define LABEL_REPRODUCTION "reproduction"
00101 #define LABEL_RESULT_FILE "result_file"
00102 #define LABEL_SIMULATOR "simulator"
00103 #define LABEL_SEED "seed"
00104 #define LABEL_STEP "step"
00105 #define LABEL_SWEEP "sweep"
00106 #define LABEL_TAXICAB "taxicab"
00107 #define LABEL_TEMPLATE1 "template1"
00108 #define LABEL_TEMPLATE2 "template2"
00109 #define LABEL_TEMPLATE3 "template3"
00110 #define LABEL_TEMPLATE4 "template4"
00111 #define LABEL_TEMPLATE5 "template5"
00112 #define LABEL_TEMPLATE6 "template6"
00113 #define LABEL_TEMPLATE7 "template7"
00114 #define LABEL_TEMPLATE8 "template8"
00115 #define LABEL_THRESHOLD "threshold"
00116 #define LABEL_TOLERANCE "tolerance"
00117 #define LABEL_VARIABLE "variable"
00118 #define LABEL_VARIABLES "variables"
00119 #define LABEL_VARIABLES_FILE "variables_file"
00120 #define LABEL_WEIGHT "weight"
00121
00122 // Enumerations
00123
00128 enum INPUT_TYPE
00129 {
00130     INPUT_TYPE_XML = 0,
00131     INPUT_TYPE_JSON = 1
00132 };
00133
00134 #endif

```

4.3 experiment.c File Reference

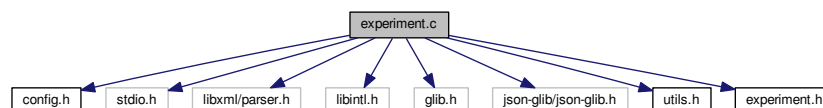
Source file to define the experiment data.

```

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

```

Include dependency graph for experiment.c:



Macros

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

Functions

- void `experiment_new` (`Experiment *experiment`)
Function to create a new `Experiment` struct.
- void `experiment_free` (`Experiment *experiment`, unsigned int type)
Function to free the memory of an `Experiment` struct.
- void `experiment_error` (`Experiment *experiment`, char *message)
Function to print a message error opening an `Experiment` struct.
- int `experiment_open_xml` (`Experiment *experiment`, xmlNode *node, unsigned int ninputs)
Function to open the `Experiment` struct on a XML node.
- int `experiment_open_json` (`Experiment *experiment`, JsonNode *node, unsigned int ninputs)
Function to open the `Experiment` struct on a XML node.

Variables

- const char * `template` [`MAX_NINPUTS`]
Array of xmlChar strings with template labels.

4.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, 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

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

Definition at line 121 of file [experiment.c](#).

```

00122 {
00123     char buffer[64];
00124     if (!experiment->name)
00125         snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00126     else
00127         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128                 experiment->name, message);
00129     error_message = g_strdup (buffer);
00130 }
```

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 88 of file [experiment.c](#).

```

00089 {
00090     unsigned int i;
00091     #if DEBUG_EXPERIMENT
00092         fprintf (stderr, "experiment_free: start\n");
00093     #endif
00094     if (type == INPUT_TYPE_XML)
00095     {
00096         for (i = 0; i < experiment->ninputs; ++i)
00097             xmlFree (experiment->template[i]);
00098         xmlFree (experiment->name);
00099     }
00100     else
00101     {
00102         for (i = 0; i < experiment->ninputs; ++i)
00103             g_free (experiment->template[i]);
00104         g_free (experiment->name);
00105     }
00106     experiment->ninputs = 0;
00107     #if DEBUG_EXPERIMENT
00108         fprintf (stderr, "experiment_free: end\n");
00109     #endif
00110 }
```

4.3.2.3 experiment_new()

```

void experiment_new (
    Experiment * experiment )
```

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

Parameters

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

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

```

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

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.

Parameters

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

Returns

1 on success, 0 on error.

Definition at line 254 of file [experiment.c](#).

```

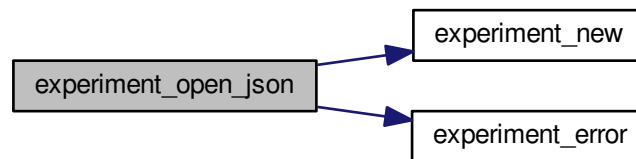
00256 {
00257     char buffer[64];
00258     JsonObject *object;
00259     const char *name;
00260     int error_code;
00261     unsigned int i;
00262
00263     #if DEBUG_EXPERIMENT
00264         fprintf (stderr, "experiment_open_json: start\n");
00265     #endif
00266
00267     // Resetting experiment data
00268     experiment\_new (experiment);
00269
00270     // Getting JSON object
00271     object = json_node_get_object (node);
00272 }
```

```

00273 // Reading the experimental data
00274 name = json_object_get_string_member (object, LABEL_NAME);
00275 if (!name)
00276 {
00277     experiment_error (experiment, _("no data file name"));
00278     goto exit_on_error;
00279 }
00280 experiment->name = g_strdup (name);
00281 #if DEBUG_EXPERIMENT
00282 fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283 #endif
00284 experiment->weight
00285 = json_object_get_float_with_default (object,
LABEL_WEIGHT, 1.,
00286                                     &error_code);
00287 if (error_code)
00288 {
00289     experiment_error (experiment, _("bad weight"));
00290     goto exit_on_error;
00291 }
00292 #if DEBUG_EXPERIMENT
00293 fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294 #endif
00295 name = json_object_get_string_member (object, template[0]);
00296 if (name)
00297 {
00298     #if DEBUG_EXPERIMENT
00299         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300                 name, template[0]);
00301     #endif
00302     ++experiment->ninputs;
00303 }
00304 else
00305 {
00306     experiment_error (experiment, _("no template"));
00307     goto exit_on_error;
00308 }
00309 experiment->template[0] = g_strdup (name);
00310 for (i = 1; i < MAX_NINPUTS; ++i)
00311 {
00312     #if DEBUG_EXPERIMENT
00313         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314     #endif
00315     if (json_object_get_member (object, template[i]))
00316     {
00317         if (ninputs && ninputs <= i)
00318         {
00319             experiment_error (experiment, _("bad templates number"));
00320             goto exit_on_error;
00321         }
00322         name = json_object_get_string_member (object, template[i]);
00323         #if DEBUG_EXPERIMENT
00324             fprintf (stderr,
00325                     "experiment_open_json: experiment=%s template%u=%s\n",
00326                     experiment->nexperiments, name, template[i]);
00327         #endif
00328         experiment->template[i] = g_strdup (name);
00329         ++experiment->ninputs;
00330     }
00331     else if (ninputs && ninputs > i)
00332     {
00333         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334         experiment_error (experiment, buffer);
00335         goto exit_on_error;
00336     }
00337     else
00338         break;
00339 }
00340 #if DEBUG_EXPERIMENT
00341 fprintf (stderr, "experiment_open_json: end\n");
00342 #endif
00343 return 1;
00344
00345 exit_on_error:
00346 experiment_free (experiment, INPUT_TYPE_JSON);
00347 #if DEBUG_EXPERIMENT
00348 fprintf (stderr, "experiment_open_json: end\n");
00349 #endif
00350 return 0;
00351 }
00352 }

```

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.

Parameters

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

Returns

1 on success, 0 on error.

Definition at line [145](#) of file [experiment.c](#).

```

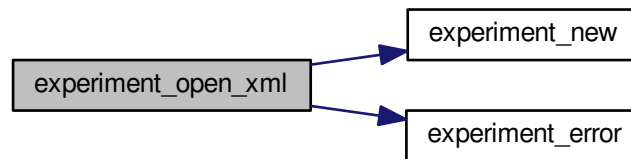
00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, _("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
  
```

```

00168 #endif
00169     experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
00172     LABEL_WEIGHT, 1.,
00173     &error_code);
00174     if (error_code)
00175     {
00176         experiment_error (experiment, _("bad weight"));
00177         goto exit_on_error;
00178     }
00179 #if DEBUG_EXPERIMENT
00180     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00181 #endif
00182     experiment->template[0]
00183     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00184     if (experiment->template[0])
00185     {
00186         #if DEBUG_EXPERIMENT
00187             fprintf (stderr, "experiment_open_xml: experiment=%s templatel=%s\n",
00188                     experiment->name, template[0]);
00189         #endif
00190         ++experiment->ninputs;
00191     }
00192     else
00193     {
00194         experiment_error (experiment, _("no template"));
00195         goto exit_on_error;
00196     }
00197     for (i = 1; i < MAX_NINPUTS; ++i)
00198     {
00199         #if DEBUG_EXPERIMENT
00200             fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00201         #endif
00202         if (xmlHasProp (node, (const xmlChar *) template[i]))
00203         {
00204             if (ninputs && ninputs <= i)
00205             {
00206                 experiment_error (experiment, _("bad templates number"));
00207                 goto exit_on_error;
00208             }
00209             experiment->template[i]
00210             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00211             #if DEBUG_EXPERIMENT
00212                 fprintf (stderr,
00213                         "experiment_open_xml: experiment=%s template%u=%s\n",
00214                         experiment->name, experiment->nexperiments,
00215                         experiment->template[i]);
00216             #endif
00217             ++experiment->ninputs;
00218         }
00219         else if (ninputs && ninputs > i)
00220         {
00221             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00222             experiment_error (experiment, buffer);
00223             goto exit_on_error;
00224         }
00225         else
00226             break;
00227     }
00228     #if DEBUG_EXPERIMENT
00229         fprintf (stderr, "experiment_open_xml: end\n");
00230     #endif
00231     return 1;
00232 }
00233 exit_on_error:
00234     experiment_free (experiment, INPUT_TYPE_XML);
00235     #if DEBUG_EXPERIMENT
00236         fprintf (stderr, "experiment_open_xml: end\n");
00237     #endif
00238     return 0;
00239 }

```


Here is the call graph for this function:



4.3.3 Variable Documentation

4.3.3.1 template

```
const char* template[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 template 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 2012-2016, 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 *template[MAX_NINPUTS] = {
00045     LABEL_TEMPLATE1, LABEL_TEMPLATE2,
00046     LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00047     LABEL_TEMPLATE5, LABEL_TEMPLATE6,
00048     LABEL_TEMPLATE7, LABEL_TEMPLATE8
00049 };
00050
00051 void
00052 experiment_new (Experiment * experiment)
00053 {
00054     unsigned int i;
00055     #if DEBUG_EXPERIMENT
00056         fprintf (stderr, "experiment_new: start\n");
00057     #endif
00058     experiment->name = NULL;
00059     experiment->ninputs = 0;
00060     for (i = 0; i < MAX_NINPUTS; ++i)
00061         experiment->template[i] = NULL;
00062     #if DEBUG_EXPERIMENT
00063         fprintf (stderr, "input_new: end\n");
00064     #endif
00065 }
00066
00067 void
00068 experiment_free (Experiment * experiment, unsigned int type)
00069 {
00070     unsigned int i;
00071     #if DEBUG_EXPERIMENT
00072         fprintf (stderr, "experiment_free: start\n");
00073     #endif
00074     if (type == INPUT_TYPE_XML)
00075     {
00076         for (i = 0; i < experiment->ninputs; ++i)
00077             xmlFree (experiment->template[i]);
00078         xmlFree (experiment->name);
00079     }
00080     else
00081     {
00082         for (i = 0; i < experiment->ninputs; ++i)
00083             g_free (experiment->template[i]);
00084         g_free (experiment->name);
00085     }
00086     experiment->ninputs = 0;
00087     #if DEBUG_EXPERIMENT
00088         fprintf (stderr, "experiment_free: end\n");
00089     #endif
00090 }
00091
00092 void
00093 experiment_error (Experiment * experiment, char *message)
00094 {
00095     char buffer[64];
00096     if (!experiment->name)
00097         snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00098     else
00099         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00100                 experiment->name, message);
00101     error_message = g_strdup (buffer);
00102 }
00103
00104 int
00105 experiment_open_xml (Experiment * experiment, xmlNode * node,
00106                     unsigned int ninputs)
00107 {

```

```

00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153     fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, _("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167     fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168     #endif
00169     experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
00172     LABEL_WEIGHT, 1.,
00173                                     &error_code);
00174     if (error_code)
00175     {
00176         experiment_error (experiment, _("bad weight"));
00177         goto exit_on_error;
00178     }
00179     #if DEBUG_EXPERIMENT
00180     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00181     #endif
00182     experiment->template[0]
00183     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00184     if (experiment->template[0])
00185     {
00186         #if DEBUG_EXPERIMENT
00187         fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00188                 experiment->name, template[0]);
00189         #endif
00190         ++experiment->ninputs;
00191     }
00192     else
00193     {
00194         experiment_error (experiment, _("no template"));
00195         goto exit_on_error;
00196     }
00197     for (i = 1; i < MAX_NINPUTS; ++i)
00198     {
00199         #if DEBUG_EXPERIMENT
00200         fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00201         #endif
00202         if (xmlHasProp (node, (const xmlChar *) template[i]))
00203         {
00204             if (ninputs && ninputs <= i)
00205             {
00206                 experiment_error (experiment, _("bad templates number"));
00207                 goto exit_on_error;
00208             }
00209             experiment->template[i]
00210             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00211             #if DEBUG_EXPERIMENT
00212             fprintf (stderr,
00213                     "experiment_open_xml: experiment=%s template%u=%s\n",
00214                     experiment->nexperiments, experiment->name,
00215                     experiment->template[i]);
00216             #endif
00217             ++experiment->ninputs;
00218         }
00219         else if (ninputs && ninputs > i)
00220         {
00221             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00222             experiment_error (experiment, buffer);
00223             goto exit_on_error;
00224         }
00225         else
00226             break;
00227     }
00228     #if DEBUG_EXPERIMENT
00229     fprintf (stderr, "experiment_open_xml: end\n");
00230     #endif
00231     return 1;
00232
00233 exit_on_error:

```

```

00234     experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236     fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238     return 0;
00239 }
00240
00241 int
00254 experiment_open_json (Experiment * experiment, JsonNode * node,
00255                      unsigned int ninputs)
00256 {
00257     char buffer[64];
00258     JsonObject *object;
00259     const char *name;
00260     int error_code;
00261     unsigned int i;
00262
00263 #if DEBUG_EXPERIMENT
00264     fprintf (stderr, "experiment_open_json: start\n");
00265 #endif
00266
00267     // Resetting experiment data
00268     experiment_new (experiment);
00269
00270     // Getting JSON object
00271     object = json_node_get_object (node);
00272
00273     // Reading the experimental data
00274     name = json_object_get_string_member (object, LABEL_NAME);
00275     if (!name)
00276     {
00277         experiment_error (experiment, _("no data file name"));
00278         goto exit_on_error;
00279     }
00280     experiment->name = g_strdup (name);
00281 #if DEBUG_EXPERIMENT
00282     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283 #endif
00284     experiment->weight
00285     = json_object_get_float_with_default (object,
00286     LABEL_WEIGHT, 1.,
00287                                         &error_code);
00288     if (error_code)
00289     {
00290         experiment_error (experiment, _("bad weight"));
00291         goto exit_on_error;
00292     }
00293 #if DEBUG_EXPERIMENT
00294     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00295 #endif
00296     name = json_object_get_string_member (object, template[0]);
00297     if (name)
00298     {
00299 #if DEBUG_EXPERIMENT
00300         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00301                 name, template[0]);
00302 #endif
00303         ++experiment->ninputs;
00304     }
00305     else
00306     {
00307         experiment_error (experiment, _("no template"));
00308         goto exit_on_error;
00309     }
00310     experiment->template[0] = g_strdup (name);
00311     for (i = 1; i < MAX_NINPUTS; ++i)
00312     {
00313 #if DEBUG_EXPERIMENT
00314         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00315 #endif
00316         if (json_object_get_member (object, template[i]))
00317         {
00318             if (ninputs && ninputs <= i)
00319             {
00320                 experiment_error (experiment, _("bad templates number"));
00321                 goto exit_on_error;
00322             }
00323             name = json_object_get_string_member (object, template[i]);
00324 #if DEBUG_EXPERIMENT
00325             fprintf (stderr,
00326                     "experiment_open_json: experiment=%s template%u=%s\n",
00327                     experiment->name, template[i]);
00328 #endif
00329             experiment->template[i] = g_strdup (name);
00330             ++experiment->ninputs;
00331         }
00332         else if (ninputs && ninputs > i)

```

```

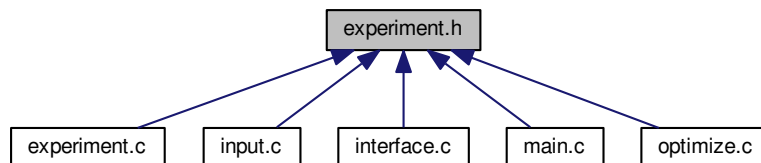
00332     {
00333         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334         experiment_error (experiment, buffer);
00335         goto exit_on_error;
00336     }
00337     else
00338         break;
00339 }
00340
00341 #if DEBUG_EXPERIMENT
00342 fprintf (stderr, "experiment_open_json: end\n");
00343 #endif
00344 return 1;
00345
00346 exit_on_error:
00347 experiment_free (experiment, INPUT_TYPE_JSON);
00348 #if DEBUG_EXPERIMENT
00349 fprintf (stderr, "experiment_open_json: end\n");
00350 #endif
00351 return 0;
00352 }

```

4.5 experiment.h File Reference

Header file to define the experiment data.

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



Data Structures

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

Functions

- void [experiment_new](#) ([Experiment](#) *experiment)
Function to create a new [Experiment](#) struct.
- void [experiment_free](#) ([Experiment](#) *experiment, unsigned int type)
Function to free the memory of an [Experiment](#) struct.
- void [experiment_error](#) ([Experiment](#) *experiment, char *message)
Function to print a message error opening an [Experiment](#) struct.
- int [experiment_open_xml](#) ([Experiment](#) *experiment, xmlDoc *node, unsigned int ninputs)
Function to open the [Experiment](#) struct on a XML node.
- int [experiment_open_json](#) ([Experiment](#) *experiment, JsonNode *node, unsigned int ninputs)
Function to open the [Experiment](#) struct on a XML node.

Variables

- `const char * template [MAX_NINPUTS]`
Array of `xmlChar` strings with template labels.

4.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, 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 [121](#) of file [experiment.c](#).

```
00122 {
00123     char buffer[64];
00124     if (!experiment->name)
00125         snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00126     else
00127         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128                 experiment->name, message);
00129     error\_message = g\_strdup (buffer);
00130 }
```

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 88 of file [experiment.c](#).

```
00089 {
00090     unsigned int i;
00091     #if DEBUG_EXPERIMENT
00092     fprintf (stderr, "experiment_free: start\n");
00093     #endif
00094     if (type == INPUT_TYPE_XML)
00095     {
00096         for (i = 0; i < experiment->ninputs; ++i)
00097             xmlFree (experiment->template[i]);
00098         xmlFree (experiment->name);
00099     }
00100     else
00101     {
00102         for (i = 0; i < experiment->ninputs; ++i)
00103             g_free (experiment->template[i]);
00104         g_free (experiment->name);
00105     }
00106     experiment->ninputs = 0;
00107     #if DEBUG_EXPERIMENT
00108     fprintf (stderr, "experiment_free: end\n");
00109     #endif
00110 }
```

4.5.2.3 experiment_new()

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

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

Parameters

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

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

```
00065 {
00066     unsigned int i;
00067     #if DEBUG_EXPERIMENT
00068     fprintf (stderr, "experiment_new: start\n");
00069     #endif
00070     experiment->name = NULL;
00071     experiment->ninputs = 0;
```

```

00072     for (i = 0; i < MAX_NINPUTS; ++i)
00073         experiment->template[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075     fprintf (stderr, "input_new: end\n");
00076 #endif
00077 }

```

4.5.2.4 experiment_open_json()

```

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

```

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

Parameters

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

Returns

1 on success, 0 on error.

Definition at line 254 of file [experiment.c](#).

```

00256 {
00257     char buffer[64];
00258     JsonObject *object;
00259     const char *name;
00260     int error_code;
00261     unsigned int i;
00262
00263 #if DEBUG_EXPERIMENT
00264     fprintf (stderr, "experiment_open_json: start\n");
00265 #endif
00266
00267     // Resetting experiment data
00268     experiment_new (experiment);
00269
00270     // Getting JSON object
00271     object = json_node_get_object (node);
00272
00273     // Reading the experimental data
00274     name = json_object_get_string_member (object, LABEL_NAME);
00275     if (!name)
00276     {
00277         experiment_error (experiment, _("no data file name"));
00278         goto exit_on_error;
00279     }
00280     experiment->name = g_strdup (name);
00281 #if DEBUG_EXPERIMENT
00282     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283 #endif
00284     experiment->weight
00285     = json_object_get_float_with_default (object,
00286     LABEL_WEIGHT, 1.,
00287     &error_code);
00288     if (error_code)
00289     {
00289         experiment_error (experiment, _("bad weight"));
00290         goto exit_on_error;

```

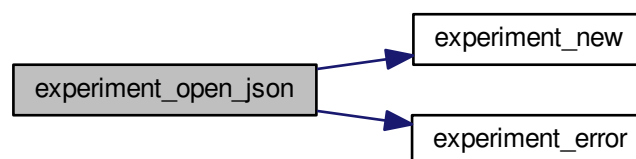


```

00291     }
00292     #if DEBUG_EXPERIMENT
00293     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294     #endif
00295     name = json_object_get_string_member (object, template[0]);
00296     if (name)
00297     {
00298     #if DEBUG_EXPERIMENT
00299     fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300             name, template[0]);
00301     #endif
00302     ++experiment->ninputs;
00303     }
00304     else
00305     {
00306     experiment_error (experiment, _("no template"));
00307     goto exit_on_error;
00308     }
00309     experiment->template[0] = g_strdup (name);
00310     for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312     #if DEBUG_EXPERIMENT
00313     fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314     #endif
00315     if (json_object_get_member (object, template[i]))
00316     {
00317     if (ninputs && ninputs <= i)
00318     {
00319     experiment_error (experiment, _("bad templates number"));
00320     goto exit_on_error;
00321     }
00322     name = json_object_get_string_member (object, template[i]);
00323     #if DEBUG_EXPERIMENT
00324     fprintf (stderr,
00325             "experiment_open_json: experiment=%s template%u=%s\n",
00326             experiment->nexperiments, name, template[i]);
00327     #endif
00328     experiment->template[i] = g_strdup (name);
00329     ++experiment->ninputs;
00330     }
00331     else if (ninputs && ninputs > i)
00332     {
00333     snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334     experiment_error (experiment, buffer);
00335     goto exit_on_error;
00336     }
00337     else
00338     break;
00339     }
00340     #if DEBUG_EXPERIMENT
00341     fprintf (stderr, "experiment_open_json: end\n");
00342     #endif
00343     return 1;
00344 }
00345
00346 exit_on_error:
00347     experiment_free (experiment, INPUT_TYPE_JSON);
00348     #if DEBUG_EXPERIMENT
00349     fprintf (stderr, "experiment_open_json: end\n");
00350     #endif
00351     return 0;
00352 }

```

Here is the call graph for this function:



4.5.2.5 experiment_open_xml()

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

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

Parameters

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

Returns

1 on success, 0 on error.

Definition at line 145 of file [experiment.c](#).

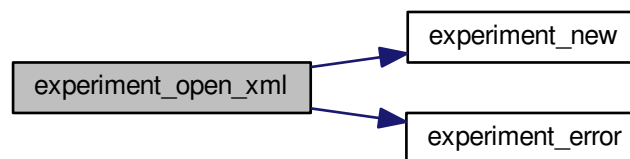
```
00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, _("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168     #endif
00169     experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
    LABEL_WEIGHT, 1.,
00172                                     &error_code);
00173     if (error_code)
00174     {
00175         experiment_error (experiment, _("bad weight"));
00176         goto exit_on_error;
00177     }
00178     #if DEBUG_EXPERIMENT
00179         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00180     #endif
00181     experiment->template[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00183     if (experiment->template[0])
00184     {
00185         #if DEBUG_EXPERIMENT
00186             fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00187                     experiment->name, template[0]);
00188         #endif
00189         ++experiment->ninputs;
00190     }
00191     else
```

```

00192     {
00193         experiment_error (experiment, _("no template"));
00194         goto exit_on_error;
00195     }
00196     for (i = 1; i < MAX_NINPUTS; ++i)
00197     {
00198 #if DEBUG_EXPERIMENT
00199         fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00200 #endif
00201         if (xmlHasProp (node, (const xmlChar *) template[i]))
00202         {
00203             if (ninputs && ninputs <= i)
00204             {
00205                 experiment_error (experiment, _("bad templates number"));
00206                 goto exit_on_error;
00207             }
00208             experiment->template[i]
00209                 = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00210 #if DEBUG_EXPERIMENT
00211             fprintf (stderr,
00212                     "experiment_open_xml: experiment=%s template%u=%s\n",
00213                     experiment->nexperiments, experiment->name,
00214                     experiment->template[i]);
00215 #endif
00216             ++experiment->ninputs;
00217         }
00218         else if (ninputs && ninputs > i)
00219         {
00220             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00221             experiment_error (experiment, buffer);
00222             goto exit_on_error;
00223         }
00224         else
00225             break;
00226     }
00227
00228 #if DEBUG_EXPERIMENT
00229     fprintf (stderr, "experiment_open_xml: end\n");
00230 #endif
00231     return 1;
00232
00233 exit_on_error:
00234     experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236     fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238     return 0;
00239 }

```

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 2012-2016, 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 *template[MAX_NINPUTS];
00039     double weight;
00040     unsigned int ninputs;
00041 } Experiment;
00042
00043 extern const char *template[MAX_NINPUTS];
00044
00045 // Public functions
00046 void experiment_new (Experiment * experiment);
00047 void experiment_free (Experiment * experiment, unsigned int type);
00048 void experiment_error (Experiment * experiment, char *message);
00049 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00050                        unsigned int ninputs);
00051 int experiment_open_json (Experiment * experiment, JsonNode * node,
00052                        unsigned int ninputs);
00053
00054 #endif

```

4.7 input.c File Reference

Source file to define the input functions.

```

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

```

Include dependency graph for input.c:



Macros

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

Functions

- `void input_new ()`
Function to create a new [Input](#) struct.
- `void input_free ()`
Function to free the memory of the input file data.
- `void input_error (char *message)`
Function to print an error message opening an [Input](#) struct.
- `int input_open_xml (xmlDoc *doc)`
Function to open the input file in XML format.
- `int input_open_json (JsonParser *parser)`
Function to open the input file in JSON format.
- `int input_open (char *filename)`
Function to open the input file.

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 2012-2016, all rights reserved.

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

4.7.2 Function Documentation

4.7.2.1 [input_error\(\)](#)

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

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

Parameters

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

Definition at line 124 of file [input.c](#).

```
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
```

4.7.2.2 input_open()

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

Function to open the input file.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 952 of file [input.c](#).

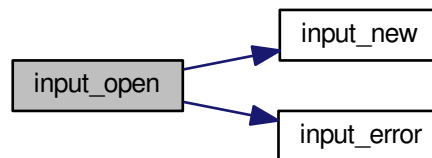
```
00953 {
00954     xmlDoc *doc;
00955     JsonParser *parser;
00956
00957     #if DEBUG_INPUT
00958     fprintf (stderr, "input_open: start\n");
00959     #endif
00960
00961     // Resetting input data
00962     input_new ();
00963
00964     // Opening input file
00965     #if DEBUG_INPUT
00966     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     return 1;
00995
00996 exit_on_error:
00997     show_error (error_message);
00998     g_free (error_message);
00999     input_free ();
01000     #if DEBUG_INPUT
01001     fprintf (stderr, "input_open: end\n");
01002     #endif
01003     return 0;
01004 }

```

Here is the call graph for this function:



4.7.2.3 input_open_json()

```

int input_open_json (
    JsonParser * parser )

```

Function to open the input file in JSON format.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 562 of file [input.c](#).

```

00563 {
00564     JsonNode *node, *child;
00565     JsonObject *object;
00566     JsonArray *array;

```

```

00567     const char *buffer;
00568     int error_code;
00569     unsigned int i, n;
00570
00571     #if DEBUG_INPUT
00572     fprintf (stderr, "input_open_json: start\n");
00573     #endif
00574
00575     // Resetting input data
00576     input->type = INPUT_TYPE_JSON;
00577
00578     // Getting the root node
00579     #if DEBUG_INPUT
00580     fprintf (stderr, "input_open_json: getting the root node\n");
00581     #endif
00582     node = json_parser_get_root (parser);
00583     object = json_node_get_object (node);
00584
00585     // Getting result and variables file names
00586     if (!input->result)
00587     {
00588         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589         if (!buffer)
00590             buffer = result_name;
00591         input->result = g_strdup (buffer);
00592     }
00593     else
00594         input->result = g_strdup (result_name);
00595     if (!input->variables)
00596     {
00597         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598         if (!buffer)
00599             buffer = variables_name;
00600         input->variables = g_strdup (buffer);
00601     }
00602     else
00603         input->variables = g_strdup (variables_name);
00604
00605     // Opening simulator program name
00606     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607     if (!buffer)
00608     {
00609         input_error (_("Bad simulator program"));
00610         goto exit_on_error;
00611     }
00612     input->simulator = g_strdup (buffer);
00613
00614     // Opening evaluator program name
00615     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616     if (buffer)
00617         input->evaluator = g_strdup (buffer);
00618
00619     // Obtaining pseudo-random numbers generator seed
00620     input->seed
00621     = json_object_get_uint_with_default (object,
00622 LABEL_SEED,
00623                                     DEFAULT_RANDOM_SEED, &error_code);
00624     if (error_code)
00625     {
00626         input_error (_("Bad pseudo-random numbers generator seed"));
00627         goto exit_on_error;
00628     }
00629
00630     // Opening algorithm
00631     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00632     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00633     {
00634         input->algorithm = ALGORITHM_MONTE_CARLO;
00635
00636         // Obtaining simulations number
00637         input->nsimulations
00638         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);
00639     };
00640     if (error_code)
00641     {
00642         input_error (_("Bad simulations number"));
00643         goto exit_on_error;
00644     }
00645     else if (!strcmp (buffer, LABEL_SWEEP))
00646         input->algorithm = ALGORITHM_SWEEP;
00647     else if (!strcmp (buffer, LABEL_GENETIC))
00648     {
00649         input->algorithm = ALGORITHM_GENETIC;
00650
00651         // Obtaining population
00652         if (json_object_get_member (object, LABEL_NPOPULATION))

```



```

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

```

```

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

```

```

00815         else
00816         {
00817             input_error
00818             (_("Unknown method to estimate the direction search"));
00819             goto exit_on_error;
00820         }
00821         input->relaxation
00822         = json_object_get_float_with_default (object,
00823 LABEL_RELAXATION,
00824                                     DEFAULT_RELAXATION,
00825                                     &error_code);
00826         if (error_code || input->relaxation < 0. || input->
00827 relaxation > 2.)
00828         {
00829             input_error (_("Invalid relaxation parameter"));
00830             goto exit_on_error;
00831         }
00832         else
00833             input->nsteps = 0;
00834         // Obtaining the threshold
00835         input->threshold
00836         = json_object_get_float_with_default (object,
00837 LABEL_THRESHOLD, 0.,
00838                                     &error_code);
00839         if (error_code)
00840         {
00841             input_error (_("Invalid threshold"));
00842             goto exit_on_error;
00843         }
00844         // Reading the experimental data
00845         array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00846         n = json_array_get_length (array);
00847         input->experiment = (Experiment *) g_malloc (n * sizeof (
00848 Experiment));
00849         for (i = 0; i < n; ++i)
00850         {
00851             #if DEBUG_INPUT
00852             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00853                     input->nexperiments);
00854             #endif
00855             child = json_array_get_element (array, i);
00856             if (!input->nexperiments)
00857             {
00858                 if (!experiment_open_json (input->experiment, child, 0))
00859                     goto exit_on_error;
00860             }
00861             else
00862             {
00863                 if (!experiment_open_json (input->experiment +
00864 input->nexperiments,
00865                                     child, input->experiment->
00866 ninputs))
00867                     goto exit_on_error;
00868             }
00869             ++input->nexperiments;
00870             #if DEBUG_INPUT
00871             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00872                     input->nexperiments);
00873             #endif
00874             if (!input->nexperiments)
00875             {
00876                 input_error (_("No optimization experiments"));
00877                 goto exit_on_error;
00878             }
00879             // Reading the variables data
00880             array = json_object_get_array_member (object, LABEL_VARIABLES);
00881             n = json_array_get_length (array);
00882             input->variable = (Variable *) g_malloc (n * sizeof (
00883 Variable));
00884             for (i = 0; i < n; ++i)
00885             {
00886                 #if DEBUG_INPUT
00887                 fprintf (stderr, "input_open_json: nvariables=%u\n", input->
00888 nvariables);
00889                 #endif
00890                 child = json_array_get_element (array, i);
00891                 if (!variable_open_json (input->variable +
00892 input->nvariables, child,
00893                                     input->algorithm, input->
00894 nsteps))
00895                     goto exit_on_error;
00896                 ++input->nvariables;

```

```

00892     }
00893     if (!input->nvariables)
00894     {
00895         input_error (_("No optimization variables"));
00896         goto exit_on_error;
00897     }
00898
00899     // Obtaining the error norm
00900     if (json_object_get_member (object, LABEL_NORM))
00901     {
00902         buffer = json_object_get_string_member (object, LABEL_NORM);
00903         if (!strcmp (buffer, LABEL_EUCLIDIAN))
00904             input->norm = ERROR_NORM_EUCLIDIAN;
00905         else if (!strcmp (buffer, LABEL_MAXIMUM))
00906             input->norm = ERROR_NORM_MAXIMUM;
00907         else if (!strcmp (buffer, LABEL_P))
00908         {
00909             input->norm = ERROR_NORM_P;
00910             input->p = json_object_get_float (object,
00911 LABEL_P, &error_code);
00912             if (!error_code)
00913             {
00914                 input_error (_("Bad P parameter"));
00915                 goto exit_on_error;
00916             }
00917         }
00918         else if (!strcmp (buffer, LABEL_TAXICAB))
00919             input->norm = ERROR_NORM_TAXICAB;
00920         else
00921         {
00922             input_error (_("Unknown error norm"));
00923             goto exit_on_error;
00924         }
00925     }
00926     else
00927         input->norm = ERROR_NORM_EUCLIDIAN;
00928
00929     // Closing the JSON document
00930     g_object_unref (parser);
00931
00932     #if DEBUG_INPUT
00933     fprintf (stderr, "input_open_json: end\n");
00934     #endif
00935     return 1;
00936
00937 exit_on_error:
00938     g_object_unref (parser);
00939     #if DEBUG_INPUT
00940     fprintf (stderr, "input_open_json: end\n");
00941     #endif
00942     return 0;
00943 }

```

Here is the call graph for this function:



4.7.2.4 input_open_xml()

```

int input_open_xml (
    xmlDoc * doc )

```

Function to open the input file in XML format.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 139 of file [input.c](#).

```

00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148     fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157     fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (_("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
00173     }
00174     if (!input->variables)
00175     {
00176         input->variables =
00177             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178         if (!input->variables)
00179             input->variables =
00180                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
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 *)
LABEL_SEED,
00199                                     DEFAULT_RANDOM_SEED, &error_code);
00200     if (error_code)
00201     {
00202         input_error (_("Bad pseudo-random numbers generator seed"));
00203         goto exit_on_error;
00204     }
00205
00206     // Opening algorithm

```

```

00207     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00208     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00209     {
00210         input->algorithm = ALGORITHM_MONTE_CARLO;
00211
00212         // Obtaining simulations number
00213         input->nsimulations
00214         = xml_node_get_int (node, (const xmlChar *)
00215 LABEL_NSIMULATIONS,
00216                             &error_code);
00217         if (error_code)
00218         {
00219             input_error (_("Bad simulations number"));
00220             goto exit_on_error;
00221         }
00222     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223         input->algorithm = ALGORITHM_SWEEP;
00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226         input->algorithm = ALGORITHM_GENETIC;
00227
00228         // Obtaining population
00229         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230         {
00231             input->nsimulations
00232             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00233                                 &error_code);
00234             if (error_code || input->nsimulations < 3)
00235             {
00236                 input_error (_("Invalid population number"));
00237                 goto exit_on_error;
00238             }
00239         }
00240     else
00241     {
00242         input_error (_("No population number"));
00243         goto exit_on_error;
00244     }
00245
00246     // Obtaining generations
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248     {
00249         input->niterations
00250         = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00251                             &error_code);
00252         if (error_code || !input->niterations)
00253         {
00254             input_error (_("Invalid generations number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         input_error (_("No generations number"));
00261         goto exit_on_error;
00262     }
00263
00264     // Obtaining mutation probability
00265     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266     {
00267         input->mutation_ratio
00268         = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00269                             &error_code);
00270         if (error_code || input->mutation_ratio < 0.
00271             || input->mutation_ratio >= 1.)
00272         {
00273             input_error (_("Invalid mutation probability"));
00274             goto exit_on_error;
00275         }
00276     }
00277     else
00278     {
00279         input_error (_("No mutation probability"));
00280         goto exit_on_error;
00281     }
00282
00283     // Obtaining reproduction probability
00284     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285     {
00286         input->reproduction_ratio
00287         = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00288                             &error_code);
00289         if (error_code || input->reproduction_ratio < 0.
00290             || input->reproduction_ratio >= 1.0)
00291         {
00292             input_error (_("Invalid reproduction probability"));

```

```

00293         goto exit_on_error;
00294     }
00295 }
00296 else
00297 {
00298     input_error (_("No reproduction probability"));
00299     goto exit_on_error;
00300 }
00301
00302 // Obtaining adaptation probability
00303 if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304 {
00305     input->adaptation_ratio
00306         = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00307                               &error_code);
00308     if (error_code || input->adaptation_ratio < 0.
00309         || input->adaptation_ratio >= 1.)
00310     {
00311         input_error (_("Invalid adaptation probability"));
00312         goto exit_on_error;
00313     }
00314 }
00315 else
00316 {
00317     input_error (_("No adaptation probability"));
00318     goto exit_on_error;
00319 }
00320
00321 // Checking survivals
00322 i = input->mutation_ratio * input->nsimulations;
00323 i += input->reproduction_ratio * input->
00324 nsimulations;
00325 i += input->adaptation_ratio * input->
00326 nsimulations;
00327 if (i > input->nsimulations - 2)
00328 {
00329     input_error
00330         (_("No enough survival entities to reproduce the population"));
00331     goto exit_on_error;
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 {
00344     // Obtaining iterations number
00345     input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
00347 LABEL_NITERATIONS,
00348                             &error_code);
00349     if (error_code == 1)
00350         input->niterations = 1;
00351     else if (error_code)
00352     {
00353         input_error (_("Bad iterations number"));
00354         goto exit_on_error;
00355     }
00356
00357     // Obtaining best number
00358     input->nbest
00359         = xml_node_get_uint_with_default (node, (const xmlChar *)
00360 LABEL_NBEST,
00361                                           1, &error_code);
00362     if (error_code || !input->nbest)
00363     {
00364         input_error (_("Invalid best number"));
00365         goto exit_on_error;
00366     }
00367     if (input->nbest > input->nsimulations)
00368     {
00369         input_error (_("Best number higher than simulations number"));
00370         goto exit_on_error;
00371     }
00372
00373     // Obtaining tolerance
00374     input->tolerance
00375         = xml_node_get_float_with_default (node,
00376                                           (const xmlChar *) LABEL_TOLERANCE,
00377                                           0., &error_code);

```

```

00376     if (error_code || input->tolerance < 0.)
00377     {
00378         input_error (_,("Invalid tolerance"));
00379         goto exit_on_error;
00380     }
00381
00382     // Getting direction search method parameters
00383     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384     {
00385         input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00387                               &error_code);
00388         if (error_code)
00389         {
00390             input_error (_,("Invalid steps number"));
00391             goto exit_on_error;
00392         }
00393         buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->direction = DIRECTION_METHOD_COORDINATES;
00396         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397         {
00398             input->direction = DIRECTION_METHOD_RANDOM;
00399             input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                                     &error_code);
00402             if (error_code || !input->nestimates)
00403             {
00404                 input_error (_,("Invalid estimates number"));
00405                 goto exit_on_error;
00406             }
00407         }
00408         else
00409         {
00410             input_error
00411                 (_,("Unknown method to estimate the direction search"));
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,
00421                                               &error_code);
00422         if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00423         {
00424             input_error (_,("Invalid relaxation parameter"));
00425             goto exit_on_error;
00426         }
00427         else
00428             input->nsteps = 0;
00429     }
00430
00431     // Obtaining the threshold
00432     input->threshold =
00433         xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00434                                         0., &error_code);
00435     if (error_code)
00436     {
00437         input_error (_,("Invalid threshold"));
00438         goto exit_on_error;
00439     }
00440
00441     // Reading the experimental data
00442     for (child = node->children; child; child = child->next)
00443     {
00444         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445             break;
00446 #if DEBUG_INPUT
00447         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00448                 input->nexperiments);
00449 #endif
00450         input->experiment = (Experiment *)
00451             g_realloc (input->experiment,
00452                       (1 + input->nexperiments) * sizeof (
Experiment));
00453         if (!input->nexperiments)
00454         {
00455             if (!experiment_open_xml (input->experiment, child, 0))
00456                 goto exit_on_error;
00457         }
00458         else

```



```

00459     {
00460         if (!experiment_open_xml (input->experiment +
input->nexperiments,
00461                                 child, input->experiment->
ninputs))
00462             goto exit_on_error;
00463     }
00464     ++input->nexperiments;
00465     #if DEBUG_INPUT
00466     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
input->nexperiments);
00467     #endif
00468     }
00469     if (!input->nexperiments)
00470     {
00471         input_error (_("No optimization experiments"));
00472         goto exit_on_error;
00473     }
00474     }
00475     buffer = NULL;
00476     // Reading the variables data
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",
00485                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00486             input_error (buffer2);
00487             goto exit_on_error;
00488         }
00489         input->variable = (Variable *)
00490             g_realloc (input->variable,
00491                     (1 + input->nvariables) * sizeof (Variable));
00492         if (!variable_open_xml (input->variable +
input->nvariables, child,
00493                               input->algorithm, input->nsteps))
00494             goto exit_on_error;
00495         ++input->nvariables;
00496     }
00497     if (!input->nvariables)
00498     {
00499         input_error (_("No optimization variables"));
00500         goto exit_on_error;
00501     }
00502     buffer = NULL;
00503     // Obtaining the error norm
00504     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00505     {
00506         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00507         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00508             input->norm = ERROR_NORM_EUCLIDIAN;
00509         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00510             input->norm = ERROR_NORM_MAXIMUM;
00511         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00512         {
00513             input->norm = ERROR_NORM_P;
00514             input->p
00515                 =
00516                 xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00517             if (!error_code)
00518             {
00519                 input_error (_("Bad P parameter"));
00520                 goto exit_on_error;
00521             }
00522         }
00523         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00524             input->norm = ERROR_NORM_TAXICAB;
00525         else
00526         {
00527             input_error (_("Unknown error norm"));
00528             goto exit_on_error;
00529         }
00530     }
00531     xmlFree (buffer);
00532     }
00533     else
00534         input->norm = ERROR_NORM_EUCLIDIAN;
00535     // Closing the XML document
00536     xmlFreeDoc (doc);
00537     #if DEBUG_INPUT
00538     fprintf (stderr, "input_open_xml: end\n");
00539     #endif

```

```

00543     return 1;
00544
00545 exit_on_error:
00546     xmlFree (buffer);
00547     xmlFreeDoc (doc);
00548     #if DEBUG_INPUT
00549     fprintf (stderr, "input_open_xml: end\n");
00550     #endif
00551     return 0;
00552 }

```

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 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <glib/gstdio.h>
00040 #include <json-glib/json-glib.h>
00041 #include "utils.h"
00042 #include "experiment.h"
00043 #include "variable.h"
00044 #include "input.h"
00045
00046 #define DEBUG_INPUT 0
00047
00048 Input input[1];
00049

```

```

00056 const char *result_name = "result";
00057 const char *variables_name = "variables";
00058
00063 void
00064 input_new ()
00065 {
00066     #if DEBUG_INPUT
00067         fprintf (stderr, "input_new: start\n");
00068     #endif
00069     input->nvariables = input->nexperiments = input->nsteps = 0;
00070     input->simulator = input->evaluator = input->directory = input->
        name = NULL;
00071     input->experiment = NULL;
00072     input->variable = NULL;
00073     #if DEBUG_INPUT
00074         fprintf (stderr, "input_new: end\n");
00075     #endif
00076 }
00077
00082 void
00083 input_free ()
00084 {
00085     unsigned int i;
00086     #if DEBUG_INPUT
00087         fprintf (stderr, "input_free: start\n");
00088     #endif
00089     g_free (input->name);
00090     g_free (input->directory);
00091     for (i = 0; i < input->nexperiments; ++i)
00092         experiment_free (input->experiment + i, input->type);
00093     for (i = 0; i < input->nvariables; ++i)
00094         variable_free (input->variable + i, input->type);
00095     g_free (input->experiment);
00096     g_free (input->variable);
00097     if (input->type == INPUT_TYPE_XML)
00098     {
00099         xmlFree (input->evaluator);
00100         xmlFree (input->simulator);
00101         xmlFree (input->result);
00102         xmlFree (input->variables);
00103     }
00104     else
00105     {
00106         g_free (input->evaluator);
00107         g_free (input->simulator);
00108         g_free (input->result);
00109         g_free (input->variables);
00110     }
00111     input->nexperiments = input->nvariables = input->nsteps = 0;
00112     #if DEBUG_INPUT
00113         fprintf (stderr, "input_free: end\n");
00114     #endif
00115 }
00116
00123 void
00124 input_error (char *message)
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
00130
00138 int
00139 input_open_xml (xmlDoc * doc)
00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148         fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150     // Resetting input data
00151     buffer = NULL;
00152     input->type = INPUT_TYPE_XML;
00153
00154     // Getting the root node
00155     #if DEBUG_INPUT
00156         fprintf (stderr, "input_open_xml: getting the root node\n");
00157     #endif
00158     node = xmlDocGetRootElement (doc);
00159     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00160     {
00161         input_error (_("Bad root XML node"));
00162     }

```

```

00163     goto exit_on_error;
00164 }
00165
00166 // Getting result and variables file names
00167 if (!input->result)
00168 {
00169     input->result =
00170         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171     if (!input->result)
00172         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173 }
00174 if (!input->variables)
00175 {
00176     input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178     if (!input->variables)
00179         input->variables =
00180             (char *) xmlStrdup ((const xmlChar *) variables_name);
00181 }
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);
00209 if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00210 {
00211     input->algorithm = ALGORITHM_MONTE_CARLO;
00212
00213     // Obtaining simulations number
00214     input->nsimulations
00215         = xml_node_get_int (node, (const xmlChar *)
00216 LABEL_NSIMULATIONS,
00217                             &error_code);
00218     if (error_code)
00219     {
00220         input_error (_("Bad simulations number"));
00221         goto exit_on_error;
00222     }
00223 }
00224 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00225     input->algorithm = ALGORITHM_SWEEP;
00226 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00227 {
00228     input->algorithm = ALGORITHM_GENETIC;
00229
00230     // Obtaining population
00231     if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00232     {
00233         input->nsimulations
00234             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00235                                 &error_code);
00236         if (error_code || input->nsimulations < 3)
00237         {
00238             input_error (_("Invalid population number"));
00239             goto exit_on_error;
00240         }
00241     }
00242     else
00243     {
00244         input_error (_("No population number"));
00245         goto exit_on_error;
00246     }
00247 }
00248
00249 // Obtaining generations
00250 if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))

```

```

00248     {
00249         input->niterations
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00251                                 &error_code);
00252         if (error_code || !input->niterations)
00253         {
00254             input_error (_("Invalid generations number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         input_error (_("No generations number"));
00261         goto exit_on_error;
00262     }
00263
00264     // Obtaining mutation probability
00265     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266     {
00267         input->mutation_ratio
00268             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00269                                 &error_code);
00270         if (error_code || input->mutation_ratio < 0.
00271             || input->mutation_ratio >= 1.)
00272         {
00273             input_error (_("Invalid mutation probability"));
00274             goto exit_on_error;
00275         }
00276     }
00277     else
00278     {
00279         input_error (_("No mutation probability"));
00280         goto exit_on_error;
00281     }
00282
00283     // Obtaining reproduction probability
00284     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285     {
00286         input->reproduction_ratio
00287             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00288                                 &error_code);
00289         if (error_code || input->reproduction_ratio < 0.
00290             || input->reproduction_ratio >= 1.0)
00291         {
00292             input_error (_("Invalid reproduction probability"));
00293             goto exit_on_error;
00294         }
00295     }
00296     else
00297     {
00298         input_error (_("No reproduction probability"));
00299         goto exit_on_error;
00300     }
00301
00302     // Obtaining adaptation probability
00303     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304     {
00305         input->adaptation_ratio
00306             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00307                                 &error_code);
00308         if (error_code || input->adaptation_ratio < 0.
00309             || input->adaptation_ratio >= 1.)
00310         {
00311             input_error (_("Invalid adaptation probability"));
00312             goto exit_on_error;
00313         }
00314     }
00315     else
00316     {
00317         input_error (_("No adaptation probability"));
00318         goto exit_on_error;
00319     }
00320
00321     // Checking survivals
00322     i = input->mutation_ratio * input->nsimulations;
00323     i += input->reproduction_ratio * input->nsimulations;
00324     i += input->adaptation_ratio * input->nsimulations;
00325     if (i > input->nsimulations - 2)
00326     {
00327         input_error
00328             (_("No enough survival entities to reproduce the population"));
00329         goto exit_on_error;
00330     }
00331 }
00332 else
00333 {
00334     input_error (_("Unknown algorithm"));

```

```

00335         goto exit_on_error;
00336     }
00337     xmlFree (buffer);
00338     buffer = NULL;
00339
00340     if (input->algorithm == ALGORITHM_MONTE_CARLO
00341         || input->algorithm == ALGORITHM_SWEEP)
00342     {
00343
00344         // Obtaining iterations number
00345         input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
00347 LABEL_NITERATIONS,
00348                             &error_code);
00349         if (error_code == 1)
00350             input->niterations = 1;
00351         else if (error_code)
00352         {
00353             input_error (_("Bad iterations number"));
00354             goto exit_on_error;
00355         }
00356
00357         // Obtaining best number
00358         input->nbest
00359         = xml_node_get_uint_with_default (node, (const xmlChar *)
00360 LABEL_NBEST,
00361                                         1, &error_code);
00362         if (error_code || !input->nbest)
00363         {
00364             input_error (_("Invalid best number"));
00365             goto exit_on_error;
00366         }
00367         if (input->nbest > input->nsimulations)
00368         {
00369             input_error (_("Best number higher than simulations number"));
00370             goto exit_on_error;
00371         }
00372
00373         // Obtaining tolerance
00374         input->tolerance
00375         = xml_node_get_float_with_default (node,
00376                                         (const xmlChar *) LABEL_TOLERANCE,
00377                                         0., &error_code);
00378         if (error_code || input->tolerance < 0.)
00379         {
00380             input_error (_("Invalid tolerance"));
00381             goto exit_on_error;
00382         }
00383
00384         // Getting direction search method parameters
00385         if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00386         {
00387             input->nsteps =
00388             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00389                               &error_code);
00390             if (error_code)
00391             {
00392                 input_error (_("Invalid steps number"));
00393                 goto exit_on_error;
00394             }
00395             buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00396             if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00397                 input->direction = DIRECTION_METHOD_COORDINATES;
00398             else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00399             {
00400                 input->direction = DIRECTION_METHOD_RANDOM;
00401                 input->nestimates
00402                 = xml_node_get_uint (node, (const xmlChar *)
00403 LABEL_NESTIMATES,
00404                                     &error_code);
00405                 if (error_code || !input->nestimates)
00406                 {
00407                     input_error (_("Invalid estimates number"));
00408                     goto exit_on_error;
00409                 }
00410             }
00411             else
00412             {
00413                 input_error
00414                 (_("Unknown method to estimate the direction search"));
00415                 goto exit_on_error;
00416             }
00417             xmlFree (buffer);
00418             buffer = NULL;
00419             input->relaxation
00420             = xml_node_get_float_with_default (node,
00421                                             (const xmlChar *)

```

```

00419                                     LABEL_RELAXATION,
00420                                     DEFAULT_RELAXATION,
00421                                     &error_code);
00422     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00423     {
00424         input_error (_("Invalid relaxation parameter"));
00425         goto exit_on_error;
00426     }
00427 }
00428 else
00429     input->nsteps = 0;
00430 }
00431 // Obtaining the threshold
00432 input->threshold =
00433     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00434                                     0., &error_code);
00435 if (error_code)
00436 {
00437     input_error (_("Invalid threshold"));
00438     goto exit_on_error;
00439 }
00440
00441 // Reading the experimental data
00442 for (child = node->children; child; child = child->next)
00443 {
00444     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445         break;
00446 #if DEBUG_INPUT
00447     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
input->nexperiments);
00448 #endif
00449     input->experiment = (Experiment *)
00450         g_realloc (input->experiment,
00451                   (1 + input->nexperiments) * sizeof (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,
00460                                 child, input->experiment->ninputs))
00461             goto exit_on_error;
00462     }
00463     ++input->nexperiments;
00464 #if DEBUG_INPUT
00465     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
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 for (; child; child = child->next)
00477 {
00478     #if DEBUG_INPUT
00479     fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00480     #endif
00481     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00482     {
00483         snprintf (buffer2, 64, "%s %u: %s",
00484                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00485         input_error (buffer2);
00486         goto exit_on_error;
00487     }
00488     input->variable = (Variable *)
00489         g_realloc (input->variable,
00490                   (1 + input->nvariables) * sizeof (Variable));
00491     if (!variable_open_xml (input->variable + input->
nvariables, child,
00492                             input->algorithm, input->nsteps))
00493         goto exit_on_error;
00494     ++input->nvariables;
00495 }
00496 if (!input->nvariables)
00497 {
00498     input_error (_("No optimization variables"));
00499     goto exit_on_error;
00500 }
00501

```

```

00502     }
00503     buffer = NULL;
00504
00505     // Obtaining the error norm
00506     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507     {
00508         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00509         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510             input->norm = ERROR_NORM_EUCLIDIAN;
00511         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512             input->norm = ERROR_NORM_MAXIMUM;
00513         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514         {
00515             input->norm = ERROR_NORM_P;
00516             input->p
00517             =
00518             xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00519             if (!error_code)
00520             {
00521                 input_error (_("Bad P parameter"));
00522                 goto exit_on_error;
00523             }
00524         }
00525         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526             input->norm = ERROR_NORM_TAXICAB;
00527         else
00528         {
00529             input_error (_("Unknown error norm"));
00530             goto exit_on_error;
00531         }
00532         xmlFree (buffer);
00533     }
00534     else
00535         input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537     // Closing the XML document
00538     xmlFreeDoc (doc);
00539
00540     #if DEBUG_INPUT
00541     fprintf (stderr, "input_open_xml: end\n");
00542     #endif
00543     return 1;
00544
00545 exit_on_error:
00546     xmlFree (buffer);
00547     xmlFreeDoc (doc);
00548     #if DEBUG_INPUT
00549     fprintf (stderr, "input_open_xml: end\n");
00550     #endif
00551     return 0;
00552 }
00553
00561 int
00562 input_open_json (JsonParser * parser)
00563 {
00564     JsonNode *node, *child;
00565     JsonObject *object;
00566     JsonArray *array;
00567     const char *buffer;
00568     int error_code;
00569     unsigned int i, n;
00570
00571     #if DEBUG_INPUT
00572     fprintf (stderr, "input_open_json: start\n");
00573     #endif
00574
00575     // Resetting input data
00576     input->type = INPUT_TYPE_JSON;
00577
00578     // Getting the root node
00579     #if DEBUG_INPUT
00580     fprintf (stderr, "input_open_json: getting the root node\n");
00581     #endif
00582     node = json_parser_get_root (parser);
00583     object = json_node_get_object (node);
00584
00585     // Getting result and variables file names
00586     if (!input->result)
00587     {
00588         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589         if (!buffer)
00590             buffer = result_name;
00591         input->result = g_strdup (buffer);
00592     }
00593     else
00594         input->result = g_strdup (result_name);
00595     if (!input->variables)

```



```

00596     {
00597         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598         if (!buffer)
00599             buffer = variables_name;
00600         input->variables = g_strdup (buffer);
00601     }
00602     else
00603         input->variables = g_strdup (variables_name);
00604
00605     // Opening simulator program name
00606     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607     if (!buffer)
00608     {
00609         input_error (_("Bad simulator program"));
00610         goto exit_on_error;
00611     }
00612     input->simulator = g_strdup (buffer);
00613
00614     // Opening evaluator program name
00615     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616     if (buffer)
00617         input->evaluator = g_strdup (buffer);
00618
00619     // Obtaining pseudo-random numbers generator seed
00620     input->seed
00621     = json_object_get_uint_with_default (object,
00622     LABEL_SEED,
00623     DEFAULT_RANDOM_SEED, &error_code);
00624     if (error_code)
00625     {
00626         input_error (_("Bad pseudo-random numbers generator seed"));
00627         goto exit_on_error;
00628     }
00629
00630     // Opening algorithm
00631     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00632     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00633     {
00634         input->algorithm = ALGORITHM_MONTE_CARLO;
00635
00636         // Obtaining simulations number
00637         input->nsimulations
00638         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00639 );
00640         if (error_code)
00641         {
00642             input_error (_("Bad simulations number"));
00643             goto exit_on_error;
00644         }
00645     }
00646     else if (!strcmp (buffer, LABEL_SWEEP))
00647         input->algorithm = ALGORITHM_SWEEP;
00648     else if (!strcmp (buffer, LABEL_GENETIC))
00649     {
00650         input->algorithm = ALGORITHM_GENETIC;
00651
00652         // Obtaining population
00653         if (json_object_get_member (object, LABEL_NPOPULATION))
00654         {
00655             input->nsimulations
00656             = json_object_get_uint (object,
00657             LABEL_NPOPULATION, &error_code);
00658             if (error_code || input->nsimulations < 3)
00659             {
00660                 input_error (_("Invalid population number"));
00661                 goto exit_on_error;
00662             }
00663         }
00664     }
00665     else
00666     {
00667         input_error (_("No population number"));
00668         goto exit_on_error;
00669     }
00670
00671     // Obtaining generations
00672     if (json_object_get_member (object, LABEL_NGENERATIONS))
00673     {
00674         input->niterations
00675         = json_object_get_uint (object,
00676         LABEL_NGENERATIONS, &error_code);
00677         if (error_code || !input->niterations)
00678         {
00679             input_error (_("Invalid generations number"));
00680             goto exit_on_error;
00681         }
00682     }
00683     else

```

```

00679     {
00680         input_error (_("No generations number"));
00681         goto exit_on_error;
00682     }
00683
00684     // Obtaining mutation probability
00685     if (json_object_get_member (object, LABEL_MUTATION))
00686     {
00687         input->mutation_ratio
00688         = json_object_get_float (object, LABEL_MUTATION, &error_code
00689 );
00689         if (error_code || input->mutation_ratio < 0.
00690             || input->mutation_ratio >= 1.)
00691         {
00692             input_error (_("Invalid mutation probability"));
00693             goto exit_on_error;
00694         }
00695     }
00696     else
00697     {
00698         input_error (_("No mutation probability"));
00699         goto exit_on_error;
00700     }
00701
00702     // Obtaining reproduction probability
00703     if (json_object_get_member (object, LABEL_REPRODUCTION))
00704     {
00705         input->reproduction_ratio
00706         = json_object_get_float (object,
00707 LABEL_REPRODUCTION, &error_code);
00707         if (error_code || input->reproduction_ratio < 0.
00708             || input->reproduction_ratio >= 1.0)
00709         {
00710             input_error (_("Invalid reproduction probability"));
00711             goto exit_on_error;
00712         }
00713     }
00714     else
00715     {
00716         input_error (_("No reproduction probability"));
00717         goto exit_on_error;
00718     }
00719
00720     // Obtaining adaptation probability
00721     if (json_object_get_member (object, LABEL_ADAPTATION))
00722     {
00723         input->adaptation_ratio
00724         = json_object_get_float (object,
00725 LABEL_ADAPTATION, &error_code);
00725         if (error_code || input->adaptation_ratio < 0.
00726             || input->adaptation_ratio >= 1.)
00727         {
00728             input_error (_("Invalid adaptation probability"));
00729             goto exit_on_error;
00730         }
00731     }
00732     else
00733     {
00734         input_error (_("No adaptation probability"));
00735         goto exit_on_error;
00736     }
00737
00738     // Checking survivals
00739     i = input->mutation_ratio * input->nsimulations;
00740     i += input->reproduction_ratio * input->nsimulations;
00741     i += input->adaptation_ratio * input->nsimulations;
00742     if (i > input->nsimulations - 2)
00743     {
00744         input_error
00745         (_("No enough survival entities to reproduce the population"));
00746         goto exit_on_error;
00747     }
00748 }
00749 else
00750 {
00751     input_error (_("Unknown algorithm"));
00752     goto exit_on_error;
00753 }
00754
00755 if (input->algorithm == ALGORITHM_MONTE_CARLO
00756     || input->algorithm == ALGORITHM_SWEEP)
00757 {
00758
00759     // Obtaining iterations number
00760     input->niterations
00761     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00762 );

```

```

00762     if (error_code == 1)
00763         input->niterations = 1;
00764     else if (error_code)
00765     {
00766         input_error (_("Bad iterations number"));
00767         goto exit_on_error;
00768     }
00769
00770     // Obtaining best number
00771     input->nbest
00772     = json_object_get_uint_with_default (object,
00773 LABEL_NBEST, 1,
00774                                         &error_code);
00775     if (error_code || !input->nbest)
00776     {
00777         input_error (_("Invalid best number"));
00778         goto exit_on_error;
00779     }
00780
00781     // Obtaining tolerance
00782     input->tolerance
00783     = json_object_get_float_with_default (object,
00784 LABEL_TOLERANCE, 0.,
00785                                         &error_code);
00786     if (error_code || input->tolerance < 0.)
00787     {
00788         input_error (_("Invalid tolerance"));
00789         goto exit_on_error;
00790     }
00791
00792     // Getting direction search method parameters
00793     if (json_object_get_member (object, LABEL_NSTEPS))
00794     {
00795         input->nsteps
00796         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00797         if (error_code)
00798         {
00799             input_error (_("Invalid steps number"));
00800             goto exit_on_error;
00801         }
00802         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00803         if (!strcmp (buffer, LABEL_COORDINATES))
00804             input->direction = DIRECTION_METHOD_COORDINATES;
00805         else if (!strcmp (buffer, LABEL_RANDOM))
00806         {
00807             input->direction = DIRECTION_METHOD_RANDOM;
00808             input->nestimates
00809             =
00810             json_object_get_uint (object,
00811 LABEL_NESTIMATES, &error_code);
00812             if (error_code || !input->nestimates)
00813             {
00814                 input_error (_("Invalid estimates number"));
00815                 goto exit_on_error;
00816             }
00817         }
00818         else
00819         {
00820             input_error
00821             (_("Unknown method to estimate the direction search"));
00822             goto exit_on_error;
00823         }
00824         input->relaxation
00825         = json_object_get_float_with_default (object,
00826 LABEL_RELAXATION,
00827                                         DEFAULT_RELAXATION,
00828                                         &error_code);
00829         if (error_code || input->relaxation < 0. || input->
00830 relaxation > 2.)
00831         {
00832             input_error (_("Invalid relaxation parameter"));
00833             goto exit_on_error;
00834         }
00835     }
00836     else
00837     {
00838         input->nsteps = 0;
00839     }
00840
00841     // Obtaining the threshold
00842     input->threshold
00843     = json_object_get_float_with_default (object,
00844 LABEL_THRESHOLD, 0.,
00845                                         &error_code);
00846     if (error_code)
00847     {
00848         input_error (_("Invalid threshold"));
00849         goto exit_on_error;
00850     }

```

```

00843
00844 // Reading the experimental data
00845 array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00846 n = json_array_get_length (array);
00847 input->experiment = (Experiment *) g_malloc (n * sizeof (
Experiment));
00848 for (i = 0; i < n; ++i)
00849 {
00850 #if DEBUG_INPUT
00851     fprintf (stderr, "input_open_json: nexperiments=%u\n",
00852             input->nexperiments);
00853 #endif
00854     child = json_array_get_element (array, i);
00855     if (!input->nexperiments)
00856     {
00857         if (!experiment_open_json (input->experiment, child, 0))
00858             goto exit_on_error;
00859     }
00860     else
00861     {
00862         if (!experiment_open_json (input->experiment + input->
nexperiments,
00863                                     child, input->experiment->ninputs))
00864             goto exit_on_error;
00865     }
00866     ++input->nexperiments;
00867 #if DEBUG_INPUT
00868     fprintf (stderr, "input_open_json: nexperiments=%u\n",
00869             input->nexperiments);
00870 #endif
00871 }
00872 if (!input->nexperiments)
00873 {
00874     input_error (_("No optimization experiments"));
00875     goto exit_on_error;
00876 }
00877
00878 // Reading the variables data
00879 array = json_object_get_array_member (object, LABEL_VARIABLES);
00880 n = json_array_get_length (array);
00881 input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00882 for (i = 0; i < n; ++i)
00883 {
00884 #if DEBUG_INPUT
00885     fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00886 #endif
00887     child = json_array_get_element (array, i);
00888     if (!variable_open_json (input->variable + input->
nvariables, child,
00889                             input->algorithm, input->nsteps))
00890         goto exit_on_error;
00891     ++input->nvariables;
00892 }
00893 if (!input->nvariables)
00894 {
00895     input_error (_("No optimization variables"));
00896     goto exit_on_error;
00897 }
00898
00899 // Obtaining the error norm
00900 if (json_object_get_member (object, LABEL_NORM))
00901 {
00902     buffer = json_object_get_string_member (object, LABEL_NORM);
00903     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00904         input->norm = ERROR_NORM_EUCLIDIAN;
00905     else if (!strcmp (buffer, LABEL_MAXIMUM))
00906         input->norm = ERROR_NORM_MAXIMUM;
00907     else if (!strcmp (buffer, LABEL_P))
00908     {
00909         input->norm = ERROR_NORM_P;
00910         input->p = json_object_get_float (object,
LABEL_P, &error_code);
00911         if (!error_code)
00912         {
00913             input_error (_("Bad P parameter"));
00914             goto exit_on_error;
00915         }
00916     }
00917     else if (!strcmp (buffer, LABEL_TAXICAB))
00918         input->norm = ERROR_NORM_TAXICAB;
00919     else
00920     {
00921         input_error (_("Unknown error norm"));
00922         goto exit_on_error;
00923     }
00924 }
00925 else

```

```

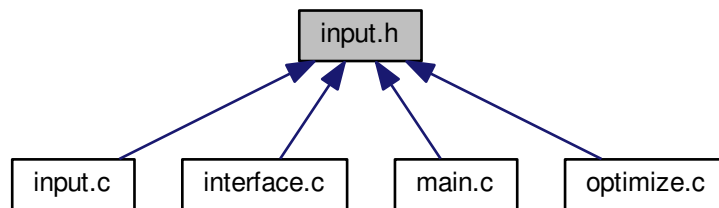
00926     input->norm = ERROR_NORM_EUCLIDIAN;
00927
00928     // Closing the JSON document
00929     g_object_unref (parser);
00930
00931     #if DEBUG_INPUT
00932     fprintf (stderr, "input_open_json: end\n");
00933     #endif
00934     return 1;
00935
00936 exit_on_error:
00937     g_object_unref (parser);
00938     #if DEBUG_INPUT
00939     fprintf (stderr, "input_open_json: end\n");
00940     #endif
00941     return 0;
00942 }
00943
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     return 1;
00995
00996 exit_on_error:
00997     show_error (error_message);
00998     g_free (error_message);
00999     input_free ();
01000     #if DEBUG_INPUT
01001     fprintf (stderr, "input_open: end\n");
01002     #endif
01003     return 0;
01004 }

```

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 [DirectionMethod](#) { [DIRECTION_METHOD_COORDINATES](#) = 0, [DIRECTION_METHOD_RANDOM](#) = 1 }
- Enum to define the methods to estimate the direction search.
- 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](#) ()
Function to create a new [Input](#) struct.
- void [input_free](#) ()
Function to free the memory of the input file data.
- void [input_error](#) (char *message)
Function to print an error message opening an [Input](#) struct.
- int [input_open_xml](#) (xmlDoc *doc)
Function to open the input file in XML format.
- int [input_open_json](#) (JsonParser *parser)
Function to open the input file in JSON format.
- int [input_open](#) (char *filename)
Function to open the input file.

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 2012-2016, all rights reserved.

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

4.9.2 Enumeration Type Documentation

4.9.2.1 DirectionMethod

enum [DirectionMethod](#)

Enum to define the methods to estimate the direction search.

Enumerator

DIRECTION_METHOD_COORDINATES	Coordinates descent method.
DIRECTION_METHOD_RANDOM	Random method.

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

```
00046 {
00047     DIRECTION_METHOD_COORDINATES = 0,
00048     DIRECTION_METHOD_RANDOM = 1,
00049 };
```

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 $.
Generated by Doxygen 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 55 of file [input.h](#).

```
00056 {
00057     ERROR_NORM_EUCLIDIAN = 0,
00059     ERROR_NORM_MAXIMUM = 1,
00061     ERROR_NORM_P = 2,
00063     ERROR_NORM_TAXICAB = 3
00065 };
```

4.9.3 Function Documentation

4.9.3.1 input_error()

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

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

Parameters

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

Definition at line 124 of file [input.c](#).

```
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
```

4.9.3.2 input_open()

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

Function to open the input file.

Parameters

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

Returns

1_on_success, 0_on_error.

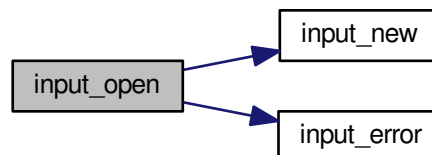
Definition at line 952 of file [input.c](#).


```

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     return 1;
00995
00996 exit_on_error:
00997     show_error (error_message);
00998     g_free (error_message);
00999     input_free ();
01000     #if DEBUG_INPUT
01001     fprintf (stderr, "input_open: end\n");
01002     #endif
01003     return 0;
01004 }

```

Here is the call graph for this function:



4.9.3.3 input_open_json()

```

int input_open_json (
    JsonParser * parser )

```

Function to open the input file in JSON format.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 562 of file [input.c](#).

```

00563 {
00564     JsonNode *node, *child;
00565     JsonObject *object;
00566     JsonArray *array;
00567     const char *buffer;
00568     int error_code;
00569     unsigned int i, n;
00570
00571     #if DEBUG_INPUT
00572     fprintf (stderr, "input_open_json: start\n");
00573     #endif
00574
00575     // Resetting input data
00576     input->type = INPUT_TYPE_JSON;
00577
00578     // Getting the root node
00579     #if DEBUG_INPUT
00580     fprintf (stderr, "input_open_json: getting the root node\n");
00581     #endif
00582     node = json_parser_get_root (parser);
00583     object = json_node_get_object (node);
00584
00585     // Getting result and variables file names
00586     if (!input->result)
00587     {
00588         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589         if (!buffer)
00590             buffer = result_name;
00591         input->result = g_strdup (buffer);
00592     }
00593     else
00594         input->result = g_strdup (result_name);
00595     if (!input->variables)
00596     {
00597         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598         if (!buffer)
00599             buffer = variables_name;
00600         input->variables = g_strdup (buffer);
00601     }
00602     else
00603         input->variables = g_strdup (variables_name);
00604
00605     // Opening simulator program name
00606     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607     if (!buffer)
00608     {
00609         input_error (_("Bad simulator program"));
00610         goto exit_on_error;
00611     }
00612     input->simulator = g_strdup (buffer);
00613
00614     // Opening evaluator program name
00615     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616     if (buffer)
00617         input->evaluator = g_strdup (buffer);
00618
00619     // Obtaining pseudo-random numbers generator seed
00620     input->seed
00621     = json_object_get_uint_with_default (object,
00622     LABEL_SEED,
00623                                     DEFAULT_RANDOM_SEED, &error_code);
00624     if (error_code)
00625     {
00626         input_error (_("Bad pseudo-random numbers generator seed"));
00627         goto exit_on_error;
00628     }

```

```

00628
00629 // Opening algorithm
00630 buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00631 if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632 {
00633     input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635     // Obtaining simulations number
00636     input->nsimulations
00637         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00638 );
00639     if (error_code)
00640     {
00641         input_error (_("Bad simulations number"));
00642         goto exit_on_error;
00643     }
00644     else if (!strcmp (buffer, LABEL_SWEEP))
00645         input->algorithm = ALGORITHM_SWEEP;
00646     else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648         input->algorithm = ALGORITHM_GENETIC;
00649
00650         // Obtaining population
00651         if (json_object_get_member (object, LABEL_NPOPULATION))
00652         {
00653             input->nsimulations
00654                 = json_object_get_uint (object,
00655 LABEL_NPOPULATION, &error_code);
00656             if (error_code || input->nsimulations < 3)
00657             {
00658                 input_error (_("Invalid population number"));
00659                 goto exit_on_error;
00660             }
00661         }
00662         else
00663         {
00664             input_error (_("No population number"));
00665             goto exit_on_error;
00666         }
00667
00668         // Obtaining generations
00669         if (json_object_get_member (object, LABEL_NGENERATIONS))
00670         {
00671             input->niterations
00672                 = json_object_get_uint (object,
00673 LABEL_NGENERATIONS, &error_code);
00674             if (error_code || !input->niterations)
00675             {
00676                 input_error (_("Invalid generations number"));
00677                 goto exit_on_error;
00678             }
00679         }
00680         else
00681         {
00682             input_error (_("No generations number"));
00683             goto exit_on_error;
00684         }
00685
00686         // Obtaining mutation probability
00687         if (json_object_get_member (object, LABEL_MUTATION))
00688         {
00689             input->mutation_ratio
00690                 = json_object_get_float (object, LABEL_MUTATION, &error_code
00691 );
00692             if (error_code || input->mutation_ratio < 0.
00693                 || input->mutation_ratio >= 1.)
00694             {
00695                 input_error (_("Invalid mutation probability"));
00696                 goto exit_on_error;
00697             }
00698         }
00699         else
00700         {
00701             input_error (_("No mutation probability"));
00702             goto exit_on_error;
00703         }
00704
00705         // Obtaining reproduction probability
00706         if (json_object_get_member (object, LABEL_REPRODUCTION))
00707         {
00708             input->reproduction_ratio
00709                 = json_object_get_float (object,
00710 LABEL_REPRODUCTION, &error_code);
00711             if (error_code || input->reproduction_ratio < 0.
00712                 || input->reproduction_ratio >= 1.0)
00713             {

```

```

00710         input_error (_("Invalid reproduction probability"));
00711         goto exit_on_error;
00712     }
00713 }
00714 else
00715 {
00716     input_error (_("No reproduction probability"));
00717     goto exit_on_error;
00718 }
00719
00720 // Obtaining adaptation probability
00721 if (json_object_get_member (object, LABEL_ADAPTATION))
00722 {
00723     input->adaptation_ratio
00724     = json_object_get_float (object,
00725 LABEL_ADAPTATION, &error_code);
00726     if (error_code || input->adaptation_ratio < 0.
00727         || input->adaptation_ratio >= 1.)
00728     {
00729         input_error (_("Invalid adaptation probability"));
00730         goto exit_on_error;
00731     }
00732 }
00733 else
00734 {
00735     input_error (_("No adaptation probability"));
00736     goto exit_on_error;
00737 }
00738 // Checking survivals
00739 i = input->mutation_ratio * input->nsimulations;
00740 i += input->reproduction_ratio * input->
00741 nsimulations;
00742 i += input->adaptation_ratio * input->
00743 nsimulations;
00744 if (i > input->nsimulations - 2)
00745 {
00746     input_error
00747     (_("No enough survival entities to reproduce the population"));
00748     goto exit_on_error;
00749 }
00750 else
00751 {
00752     input_error (_("Unknown algorithm"));
00753     goto exit_on_error;
00754 }
00755 if (input->algorithm == ALGORITHM_MONTE_CARLO
00756     || input->algorithm == ALGORITHM_SWEEP)
00757 {
00758     // Obtaining iterations number
00759     input->niterations
00760     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00761 );
00762     if (error_code == 1)
00763         input->niterations = 1;
00764     else if (error_code)
00765     {
00766         input_error (_("Bad iterations number"));
00767         goto exit_on_error;
00768     }
00769     // Obtaining best number
00770     input->nbest
00771     = json_object_get_uint_with_default (object,
00772 LABEL_NBEST, 1,
00773                                         &error_code);
00774     if (error_code || !input->nbest)
00775     {
00776         input_error (_("Invalid best number"));
00777         goto exit_on_error;
00778     }
00779     // Obtaining tolerance
00780     input->tolerance
00781     = json_object_get_float_with_default (object,
00782 LABEL_TOLERANCE, 0.,
00783                                         &error_code);
00784     if (error_code || input->tolerance < 0.)
00785     {
00786         input_error (_("Invalid tolerance"));
00787         goto exit_on_error;
00788     }
00789     // Getting direction search method parameters

```

```

00791     if (json_object_get_member (object, LABEL_NSTEPS))
00792     {
00793         input->nsteps
00794         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00795         if (error_code)
00796         {
00797             input_error (_("Invalid steps number"));
00798             goto exit_on_error;
00799         }
00800         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00801         if (!strcmp (buffer, LABEL_COORDINATES))
00802             input->direction = DIRECTION_METHOD_COORDINATES;
00803         else if (!strcmp (buffer, LABEL_RANDOM))
00804         {
00805             input->direction = DIRECTION_METHOD_RANDOM;
00806             input->nestimates
00807             =
00808             json_object_get_uint (object,
00809 LABEL_NESTIMATES, &error_code);
00810             if (error_code || !input->nestimates)
00811             {
00812                 input_error (_("Invalid estimates number"));
00813                 goto exit_on_error;
00814             }
00815         }
00816         else
00817         {
00818             input_error
00819             (_("Unknown method to estimate the direction search"));
00820             goto exit_on_error;
00821         }
00822         input->relaxation
00823         = json_object_get_float_with_default (object,
00824 LABEL_RELAXATION,
00825                                     DEFAULT_RELAXATION,
00826                                     &error_code);
00827         if (error_code || input->relaxation < 0. || input->
00828 relaxation > 2.)
00829         {
00830             input_error (_("Invalid relaxation parameter"));
00831             goto exit_on_error;
00832         }
00833         else
00834             input->nsteps = 0;
00835         // Obtaining the threshold
00836         input->threshold
00837         = json_object_get_float_with_default (object,
00838 LABEL_THRESHOLD, 0.,
00839                                     &error_code);
00840         if (error_code)
00841         {
00842             input_error (_("Invalid threshold"));
00843             goto exit_on_error;
00844         }
00845         // Reading the experimental data
00846         array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00847         n = json_array_get_length (array);
00848         input->experiment = (Experiment *) g_malloc (n * sizeof (
00849 Experiment));
00850         for (i = 0; i < n; ++i)
00851         {
00852             #if DEBUG_INPUT
00853             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00854                     input->nexperiments);
00855             #endif
00856             child = json_array_get_element (array, i);
00857             if (!input->nexperiments)
00858             {
00859                 if (!experiment_open_json (input->experiment, child, 0))
00860                     goto exit_on_error;
00861             }
00862             else
00863             {
00864                 if (!experiment_open_json (input->experiment +
00865 input->nexperiments,
00866                                     child, input->experiment->
00867 ninputs))
00868                     goto exit_on_error;
00869             }
00870             ++input->nexperiments;
00871             #if DEBUG_INPUT
00872             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00873                     input->nexperiments);
00874             #endif
00875         }

```

```

00871     }
00872     if (!input->nexperiments)
00873     {
00874         input_error (_("No optimization experiments"));
00875         goto exit_on_error;
00876     }
00877
00878     // Reading the variables data
00879     array = json_object_get_array_member (object, LABEL_VARIABLES);
00880     n = json_array_get_length (array);
00881     input->variable = (Variable *) g_malloc (n * sizeof (
Variable));
00882     for (i = 0; i < n; ++i)
00883     {
00884 #if DEBUG_INPUT
00885         fprintf (stderr, "input_open_json: nvariables=%u\n", input->
nvariables);
00886 #endif
00887         child = json_array_get_element (array, i);
00888         if (!variable_open_json (input->variable +
input->nvariables, child,
00889                                 input->algorithm, input->
nsteps))
00890             goto exit_on_error;
00891         ++input->nvariables;
00892     }
00893     if (!input->nvariables)
00894     {
00895         input_error (_("No optimization variables"));
00896         goto exit_on_error;
00897     }
00898
00899     // Obtaining the error norm
00900     if (json_object_get_member (object, LABEL_NORM))
00901     {
00902         buffer = json_object_get_string_member (object, LABEL_NORM);
00903         if (!strcmp (buffer, LABEL_EUCLIDIAN))
00904             input->norm = ERROR_NORM_EUCLIDIAN;
00905         else if (!strcmp (buffer, LABEL_MAXIMUM))
00906             input->norm = ERROR_NORM_MAXIMUM;
00907         else if (!strcmp (buffer, LABEL_P))
00908         {
00909             input->norm = ERROR_NORM_P;
00910             input->p = json_object_get_float (object,
LABEL_P, &error_code);
00911             if (!error_code)
00912             {
00913                 input_error (_("Bad P parameter"));
00914                 goto exit_on_error;
00915             }
00916         }
00917         else if (!strcmp (buffer, LABEL_TAXICAB))
00918             input->norm = ERROR_NORM_TAXICAB;
00919         else
00920         {
00921             input_error (_("Unknown error norm"));
00922             goto exit_on_error;
00923         }
00924     }
00925     else
00926         input->norm = ERROR_NORM_EUCLIDIAN;
00927
00928     // Closing the JSON document
00929     g_object_unref (parser);
00930
00931 #if DEBUG_INPUT
00932     fprintf (stderr, "input_open_json: end\n");
00933 #endif
00934     return 1;
00935
00936 exit_on_error:
00937     g_object_unref (parser);
00938 #if DEBUG_INPUT
00939     fprintf (stderr, "input_open_json: end\n");
00940 #endif
00941     return 0;
00942 }

```

Here is the call graph for this function:



4.9.3.4 input_open_xml()

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

Function to open the input file in XML format.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 139 of file [input.c](#).

```

00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148         fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157         fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (_("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
  
```

```

00172     input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
00173 }
00174 if (!input->variables)
00175 {
00176     input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178     if (!input->variables)
00179         input->variables =
00180             (char *) xmlStrdup ((const xmlChar *) variables_name);
00181 }
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 *)
LABEL_SEED,
00199                                     DEFAULT_RANDOM_SEED, &error_code);
00200 if (error_code)
00201 {
00202     input_error _("Bad pseudo-random numbers generator seed");
00203     goto exit_on_error;
00204 }
00205
00206 // Opening algorithm
00207 buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00208 if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00209 {
00210     input->algorithm = ALGORITHM_MONTE_CARLO;
00211 }
00212 // Obtaining simulations number
00213 input->nsimulations
00214     = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
00215                         &error_code);
00216 if (error_code)
00217 {
00218     input_error _("Bad simulations number");
00219     goto exit_on_error;
00220 }
00221 }
00222 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223     input->algorithm = ALGORITHM_SWEEP;
00224 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225 {
00226     input->algorithm = ALGORITHM_GENETIC;
00227 }
00228 // Obtaining population
00229 if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230 {
00231     input->nsimulations
00232         = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
&error_code);
00233     if (error_code || input->nsimulations < 3)
00234     {
00235         input_error _("Invalid population number");
00236         goto exit_on_error;
00237     }
00238 }
00239 }
00240 else
00241 {
00242     input_error _("No population number");
00243     goto exit_on_error;
00244 }
00245
00246 // Obtaining generations
00247 if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248 {
00249     input->niterations
00250         = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
&error_code);
00251     if (error_code || !input->niterations)
00252     {
00253         input_error _("Invalid generations number");
00254         goto exit_on_error;
00255     }

```



```

00256     }
00257 }
00258 else
00259 {
00260     input_error (_("No generations number"));
00261     goto exit_on_error;
00262 }
00263
00264 // Obtaining mutation probability
00265 if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266 {
00267     input->mutation_ratio
00268     = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00269                           &error_code);
00270     if (error_code || input->mutation_ratio < 0.
00271         || input->mutation_ratio >= 1.)
00272     {
00273         input_error (_("Invalid mutation probability"));
00274         goto exit_on_error;
00275     }
00276 }
00277 else
00278 {
00279     input_error (_("No mutation probability"));
00280     goto exit_on_error;
00281 }
00282
00283 // Obtaining reproduction probability
00284 if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285 {
00286     input->reproduction_ratio
00287     = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00288                           &error_code);
00289     if (error_code || input->reproduction_ratio < 0.
00290         || input->reproduction_ratio >= 1.0)
00291     {
00292         input_error (_("Invalid reproduction probability"));
00293         goto exit_on_error;
00294     }
00295 }
00296 else
00297 {
00298     input_error (_("No reproduction probability"));
00299     goto exit_on_error;
00300 }
00301
00302 // Obtaining adaptation probability
00303 if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304 {
00305     input->adaptation_ratio
00306     = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00307                           &error_code);
00308     if (error_code || input->adaptation_ratio < 0.
00309         || input->adaptation_ratio >= 1.)
00310     {
00311         input_error (_("Invalid adaptation probability"));
00312         goto exit_on_error;
00313     }
00314 }
00315 else
00316 {
00317     input_error (_("No adaptation probability"));
00318     goto exit_on_error;
00319 }
00320
00321 // Checking survivals
00322 i = input->mutation_ratio * input->nsimulations;
00323 i += input->reproduction_ratio * input->
00324 nsimulations;
00325 if (i > input->nsimulations - 2)
00326 {
00327     input_error
00328     (_("No enough survival entities to reproduce the population"));
00329     goto exit_on_error;
00330 }
00331 }
00332 else
00333 {
00334     input_error (_("Unknown algorithm"));
00335     goto exit_on_error;
00336 }
00337 xmlFree (buffer);
00338 buffer = NULL;
00339
00340 if (input->algorithm == ALGORITHM_MONTE_CARLO

```

```

00341     || input->algorithm == ALGORITHM_SWEEP)
00342     {
00343
00344         // Obtaining iterations number
00345         input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
00347                             &error_code);
00348         if (error_code == 1)
00349             input->niterations = 1;
00350         else if (error_code)
00351         {
00352             input_error (_("Bad iterations number"));
00353             goto exit_on_error;
00354         }
00355
00356         // Obtaining best number
00357         input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
00359                                         1, &error_code);
00360         if (error_code || !input->nbest)
00361         {
00362             input_error (_("Invalid best number"));
00363             goto exit_on_error;
00364         }
00365         if (input->nbest > input->nsimulations)
00366         {
00367             input_error (_("Best number higher than simulations number"));
00368             goto exit_on_error;
00369         }
00370
00371         // Obtaining tolerance
00372         input->tolerance
00373         = xml_node_get_float_with_default (node,
00374                                           (const xmlChar *) LABEL_TOLERANCE,
00375                                           0., &error_code);
00376         if (error_code || input->tolerance < 0.)
00377         {
00378             input_error (_("Invalid tolerance"));
00379             goto exit_on_error;
00380         }
00381
00382         // Getting direction search method parameters
00383         if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384         {
00385             input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00387                               &error_code);
00388             if (error_code)
00389             {
00390                 input_error (_("Invalid steps number"));
00391                 goto exit_on_error;
00392             }
00393             buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394             if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395                 input->direction = DIRECTION_METHOD_COORDINATES;
00396             else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397             {
00398                 input->direction = DIRECTION_METHOD_RANDOM;
00399                 input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                                     &error_code);
00402                 if (error_code || !input->nestimates)
00403                 {
00404                     input_error (_("Invalid estimates number"));
00405                     goto exit_on_error;
00406                 }
00407             }
00408             else
00409             {
00410                 input_error
00411                 (_("Unknown method to estimate the direction search"));
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,
00421                                               &error_code);
00422             if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00423             {

```

```

00424         input_error (_("Invalid relaxation parameter"));
00425         goto exit_on_error;
00426     }
00427 }
00428 else
00429     input->nsteps = 0;
00430 }
00431 // Obtaining the threshold
00432 input->threshold =
00433     xml_node_get_float_with_default (node, (const xmlChar *)
00434     LABEL_THRESHOLD,
00435     0., &error_code);
00436 if (error_code)
00437 {
00438     input_error (_("Invalid threshold"));
00439     goto exit_on_error;
00440 }
00441 // Reading the experimental data
00442 for (child = node->children; child; child = child->next)
00443 {
00444     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445         break;
00446 #if DEBUG_INPUT
00447     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00448             input->nexperiments);
00449 #endif
00450     input->experiment = (Experiment *)
00451         g_realloc (input->experiment,
00452             (1 + input->nexperiments) * sizeof (
00453             Experiment));
00454     if (!input->nexperiments)
00455     {
00456         if (!experiment_open_xml (input->experiment, child, 0))
00457             goto exit_on_error;
00458     }
00459     else
00460     {
00461         if (!experiment_open_xml (input->experiment +
00462         input->nexperiments,
00463         child, input->experiment->
00464         ninputs))
00465             goto exit_on_error;
00466     }
00467     ++input->nexperiments;
00468 #if DEBUG_INPUT
00469     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00470             input->nexperiments);
00471 #endif
00472     if (!input->nexperiments)
00473     {
00474         input_error (_("No optimization experiments"));
00475         goto exit_on_error;
00476     }
00477     buffer = NULL;
00478 // Reading the variables data
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 +
00495     input->nvariables, child,
00496     input->algorithm, input->nsteps))
00497         goto exit_on_error;
00498     ++input->nvariables;
00499 }
00500 if (!input->nvariables)
00501 {
00502     input_error (_("No optimization variables"));
00503     goto exit_on_error;
00504 }
00505 buffer = NULL;
00506 // Obtaining the error norm

```

```

00506  if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507  {
00508      buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00509      if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510          input->norm = ERROR_NORM_EUCLIDIAN;
00511      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512          input->norm = ERROR_NORM_MAXIMUM;
00513      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514      {
00515          input->norm = ERROR_NORM_P;
00516          input->p
00517              =
00518              xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00519          if (!error_code)
00520          {
00521              input_error (_("Bad P parameter"));
00522              goto exit_on_error;
00523          }
00524      }
00525      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526          input->norm = ERROR_NORM_TAXICAB;
00527      else
00528      {
00529          input_error (_("Unknown error norm"));
00530          goto exit_on_error;
00531      }
00532      xmlFree (buffer);
00533  }
00534  else
00535      input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537  // Closing the XML document
00538  xmlFreeDoc (doc);
00539
00540  #if DEBUG_INPUT
00541      fprintf (stderr, "input_open_xml: end\n");
00542  #endif
00543  return 1;
00544
00545 exit_on_error:
00546  xmlFree (buffer);
00547  xmlFreeDoc (doc);
00548  #if DEBUG_INPUT
00549      fprintf (stderr, "input_open_xml: end\n");
00550  #endif
00551  return 0;
00552 }

```

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 2012-2016, 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 DirectionMethod
00036 {
00037     DIRECTION_METHOD_COORDINATES = 0,
00038     DIRECTION_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 direction;
00073     unsigned int nestimates;
00074     unsigned int niterations;
00075     unsigned int nbest;
00076     unsigned int norm;
00077     unsigned int type;
00078 } Input;
00079
00080 extern Input input[1];
00081 extern const char *result_name;
00082 extern const char *variables_name;
00083
00084 // Public functions
00085 void input_new ();
00086 void input_free ();
00087 void input_error (char *message);
00088 int input_open_xml (xmlDoc * doc);
00089 int input_open_json (JsonParser * parser);
00090 int input_open (char *filename);
00091
00092 #endif

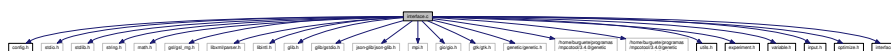
```

4.11 interface.c File Reference

Source file to define the graphical interface functions.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
```

Include dependency graph for interface.c:



Macros

- `#define DEBUG_INTERFACE 0`
Macro to debug interface functions.
- `#define INPUT_FILE "test-ga.xml"`
Macro to define the initial input file.

Functions

- void `input_save_direction_xml` (xmlNode *node)
Function to save the direction search method data in a XML node.
- void `input_save_direction_json` (JsonNode *node)
Function to save the direction search method data in a JSON node.
- void `input_save_xml` (xmlDoc *doc)
Function to save the input file in XML format.
- void `input_save_json` (JsonGenerator *generator)
Function to save the input file in JSON format.
- void `input_save` (char *filename)
Function to save the input file.
- void `options_new` ()
Function to open the options dialog.

- void [running_new](#) ()
Function to open the running dialog.
- unsigned int [window_get_algorithm](#) ()
Function to get the stochastic algorithm number.
- unsigned int [window_get_direction](#) ()
Function to get the direction search method number.
- unsigned int [window_get_norm](#) ()
Function to get the norm method number.
- void [window_save_direction](#) ()
Function to save the direction search method data in the input file.
- int [window_save](#) ()
Function to save the input file.
- void [window_run](#) ()
Function to run a optimization.
- void [window_help](#) ()
Function to show a help dialog.
- void [window_about](#) ()
Function to show an about dialog.
- void [window_update_direction](#) ()
Function to update direction search method widgets view in the main window.
- void [window_update](#) ()
Function to update the main window view.
- void [window_set_algorithm](#) ()
Function to avoid memory errors changing the algorithm.
- void [window_set_experiment](#) ()
Function to set the experiment data in the main window.
- void [window_remove_experiment](#) ()
Function to remove an experiment in the main window.
- void [window_add_experiment](#) ()
Function to add an experiment in the main window.
- void [window_name_experiment](#) ()
Function to set the experiment name in the main window.
- void [window_weight_experiment](#) ()
Function to update the experiment weight in the main window.
- void [window_inputs_experiment](#) ()
Function to update the experiment input templates number in the main window.
- void [window_template_experiment](#) (void *data)
Function to update the experiment i-th input template in the main window.
- void [window_set_variable](#) ()
Function to set the variable data in the main window.
- void [window_remove_variable](#) ()
Function to remove a variable in the main window.
- void [window_add_variable](#) ()
Function to add a variable in the main window.
- void [window_label_variable](#) ()
Function to set the variable label in the main window.
- void [window_precision_variable](#) ()
Function to update the variable precision in the main window.
- void [window_rangemin_variable](#) ()
Function to update the variable rangemin in the main window.
- void [window_rangemax_variable](#) ()

- *Function to update the variable rangemax in the main window.*
void [window_rangeminabs_variable](#) ()
- *Function to update the variable rangeminabs in the main window.*
void [window_rangemaxabs_variable](#) ()
- *Function to update the variable rangemaxabs in the main window.*
void [window_step_variable](#) ()
- *Function to update the variable step in the main window.*
void [window_update_variable](#) ()
- *Function to update the variable data in the main window.*
int [window_read](#) (char *filename)
- *Function to read the input data of a file.*
void [window_open](#) ()
- *Function to open the input data.*
void [window_new](#) (GtkApplication *application)
- *Function to open the main window.*

Variables

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

4.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.11.2 Function Documentation

4.11.2.1 input_save()

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

Function to save the input file.

Parameters

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

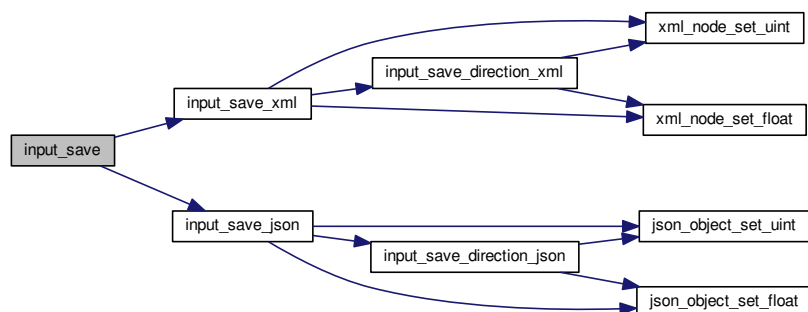
Definition at line 579 of file [interface.c](#).

```

00580 {
00581     xmlDoc *doc;
00582     JsonGenerator *generator;
00583
00584 #if DEBUG_INTERFACE
00585     fprintf (stderr, "input_save: start\n");
00586 #endif
00587
00588     // Getting the input file directory
00589     input->name = g_path_get_basename (filename);
00590     input->directory = g_path_get_dirname (filename);
00591
00592     if (input->type == INPUT_TYPE_XML)
00593     {
00594         // Opening the input file
00595         doc = xmlNewDoc ((const xmlChar *) "1.0");
00596         input_save_xml (doc);
00597
00598         // Saving the XML file
00599         xmlSaveFormatFile (filename, doc, 1);
00600
00601         // Freeing memory
00602         xmlFreeDoc (doc);
00603     }
00604     else
00605     {
00606         // Opening the input file
00607         generator = json_generator_new ();
00608         json_generator_set_pretty (generator, TRUE);
00609         input_save_json (generator);
00610
00611         // Saving the JSON file
00612         json_generator_to_file (generator, filename, NULL);
00613
00614         // Freeing memory
00615         g_object_unref (generator);
00616     }
00617
00618 #if DEBUG_INTERFACE
00619     fprintf (stderr, "input_save: end\n");
00620 #endif
00621 }

```

Here is the call graph for this function:



4.11.2.2 input_save_direction_json()

```
void input_save_direction_json (
    JsonNode * node )
```

Function to save the direction search method data in a JSON node.

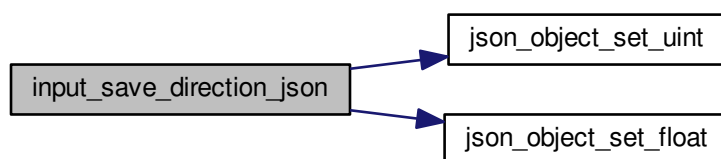
Parameters

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

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

```
00208 {
00209     JsonObject *object;
00210     #if DEBUG_INTERFACE
00211     fprintf (stderr, "input_save_direction_json: start\n");
00212     #endif
00213     object = json_node_get_object (node);
00214     if (input->nsteps)
00215     {
00216         json_object_set_uint (object, LABEL_NSTEPS,
00217                               input->nsteps);
00218         if (input->relaxation != DEFAULT_RELAXATION)
00219             json_object_set_float (object, LABEL_RELAXATION,
00220                                   input->relaxation);
00221         switch (input->direction)
00222         {
00223             case DIRECTION_METHOD_COORDINATES:
00224                 json_object_set_string_member (object, LABEL_DIRECTION,
00225                                                LABEL_COORDINATES);
00226                 break;
00227             default:
00228                 json_object_set_string_member (object, LABEL_DIRECTION,
00229                                                LABEL_RANDOM);
00230         }
00231         json_object_set_uint (object, LABEL_NESTIMATES,
00232                               input->nestimates);
00233     }
00234     #if DEBUG_INTERFACE
00235     fprintf (stderr, "input_save_direction_json: end\n");
00236     #endif
00237 }
```

Here is the call graph for this function:



4.11.2.3 input_save_direction_xml()

```
void input_save_direction_xml (
    xmlNode * node )
```

Function to save the direction search method data in a XML node.

Parameters

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

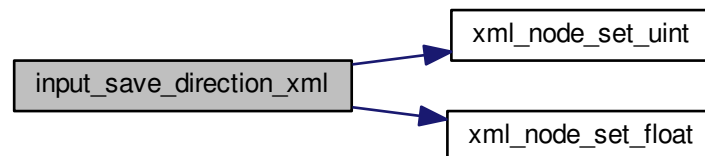
Definition at line 171 of file [interface.c](#).

```

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

```

Here is the call graph for this function:



4.11.2.4 input_save_json()

```

void input_save_json (
    JsonGenerator * generator )

```

Function to save the input file in JSON format.

Parameters

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

Definition at line 416 of file [interface.c](#).

```

00417 {
00418     unsigned int i, j;
00419     char *buffer;
00420     JsonNode *node, *child;
00421     JsonObject *object, *object2;
00422     JsonArray *array;
00423     GFile *file, *file2;
00424
00425     #if DEBUG_INTERFACE
00426         fprintf (stderr, "input_save_json: start\n");
00427     #endif
00428
00429     // Setting root JSON node
00430     node = json_node_new (JSON_NODE_OBJECT);
00431     object = json_node_get_object (node);
00432     json_generator_set_root (generator, node);
00433
00434     // Adding properties to the root JSON node
00435     if (strcmp (input->result, result_name))
00436         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00437     if (strcmp (input->variables, variables_name))
00438         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00439
00440     file = g_file_new_for_path (input->directory);
00441     file2 = g_file_new_for_path (input->simulator);
00442     buffer = g_file_get_relative_path (file, file2);
00443     g_object_unref (file2);
00444     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00445     g_free (buffer);
00446     if (input->evaluator)
00447     {
00448         file2 = g_file_new_for_path (input->evaluator);
00449         buffer = g_file_get_relative_path (file, file2);
00450         g_object_unref (file2);
00451         if (strlen (buffer))
00452             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00453         g_free (buffer);
00454     }
00455     if (input->seed != DEFAULT_RANDOM_SEED)
00456         json_object_set_uint (object, LABEL_SEED,
input->seed);
00457
00458     // Setting the algorithm
00459     buffer = (char *) g_slice_alloc (64);
00460     switch (input->algorithm)
00461     {
00462     case ALGORITHM_MONTE_CARLO:
00463         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00464         snprintf (buffer, 64, "%u", input->nsimulations);
00465         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00466         snprintf (buffer, 64, "%u", input->niterations);
00467         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00468         snprintf (buffer, 64, "%.3lg", input->tolerance);
00469         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00470         snprintf (buffer, 64, "%u", input->nbest);
00471         json_object_set_string_member (object, LABEL_NBEST, buffer);
00472         input_save_direction_json (node);
00473         break;
00474     case ALGORITHM_SWEEP:
00475         json_object_set_string_member (object, LABEL_ALGORITHM,
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_direction_json (node);
00483         break;
00484     default:
00485         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00486         snprintf (buffer, 64, "%u", input->nsimulations);
00487         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);

```

```

00489     snprintf (buffer, 64, "%u", input->niterations);
00490     json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00491     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00492     json_object_set_string_member (object, LABEL_MUTATION, buffer);
00493     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00494     json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00495     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00496     json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00497     break;
00498 }
00499 g_slice_free1 (64, buffer);
00500 if (input->threshold != 0.)
00501     json_object_set_float (object, LABEL_THRESHOLD,
00502         input->threshold);
00502
00503 // Setting the experimental data
00504 array = json_array_new ();
00505 for (i = 0; i < input->nexperiments; ++i)
00506 {
00507     child = json_node_new (JSON_NODE_OBJECT);
00508     object = json_node_get_object (child);
00509     json_object_set_string_member (object2, LABEL_NAME,
00510         input->experiment[i].name);
00511     if (input->experiment[i].weight != 1.)
00512         json_object_set_float (object2, LABEL_WEIGHT,
00513             input->experiment[i].weight);
00514     for (j = 0; j < input->experiment->ninputs; ++j)
00515         json_object_set_string_member (object2, template[j],
00516             input->experiment[i].
00517                 template[j]);
00518     json_array_add_element (array, child);
00519 }
00520 json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00521
00522 // Setting the variables data
00523 array = json_array_new ();
00524 for (i = 0; i < input->nvariables; ++i)
00525 {
00526     child = json_node_new (JSON_NODE_OBJECT);
00527     object = json_node_get_object (child);
00528     json_object_set_string_member (object2, LABEL_NAME,
00529         input->variable[i].name);
00530     json_object_set_float (object2, LABEL_MINIMUM,
00531         input->variable[i].rangemin);
00532     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00533         json_object_set_float (object2,
00534             LABEL_ABSOLUTE_MINIMUM,
00535             input->variable[i].rangeminabs);
00536     json_object_set_float (object2, LABEL_MAXIMUM,
00537         input->variable[i].rangemax);
00538     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00539         json_object_set_float (object2,
00540             LABEL_ABSOLUTE_MAXIMUM,
00541             input->variable[i].rangemaxabs);
00542     if (input->variable[i].precision !=
00543         DEFAULT_PRECISION)
00544         json_object_set_uint (object2, LABEL_PRECISION,
00545             input->variable[i].precision);
00546     if (input->algorithm == ALGORITHM_SWEEP)
00547         json_object_set_uint (object2, LABEL_NSWEEPS,
00548             input->variable[i].nsweeps);
00549     else if (input->algorithm == ALGORITHM_GENETIC)
00550         json_object_set_uint (object2, LABEL_NBITS,
00551             input->variable[i].nbits);
00552     if (input->nsteps)
00553         json_object_set_float (object, LABEL_STEP,
00554             input->variable[i].step);
00555     json_array_add_element (array, child);
00556 }
00557 json_object_set_array_member (object, LABEL_VARIABLES, array);
00558
00559 // Saving the error norm
00560 switch (input->norm)
00561 {
00562     case ERROR_NORM_MAXIMUM:
00563         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00564         break;
00565     case ERROR_NORM_P:
00566         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00567         json_object_set_float (object, LABEL_P, input->
00568             p);
00569         break;
00570     case ERROR_NORM_TAXICAB:
00571         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00572         break;
00573 }
00574 }
00575 #if DEBUG_INTERFACE

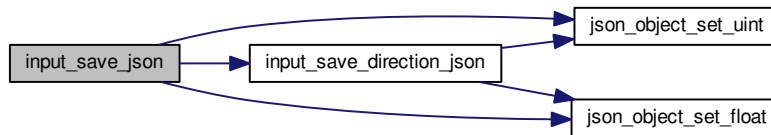
```

```

00568     fprintf (stderr, "input_save_json: end\n");
00569 #endif
00570 }

```

Here is the call graph for this function:



4.11.2.5 input_save_xml()

```

void input_save_xml (
    xmlDoc * doc )

```

Function to save the input file in XML format.

Parameters

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

Definition at line 243 of file [interface.c](#).

```

00244 {
00245     unsigned int i, j;
00246     char *buffer;
00247     xmlNode *node, *child;
00248     GFile *file, *file2;
00249
00250 #if DEBUG_INTERFACE
00251     fprintf (stderr, "input_save_xml: start\n");
00252 #endif
00253
00254     // Setting root XML node
00255     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00256     xmlDocSetRootElement (doc, node);
00257
00258     // Adding properties to the root XML node
00259     if (xmlStrcmp
00260         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00261         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00262             (xmlChar *) input->result);
00263     if (xmlStrcmp
00264         ((const xmlChar *) input->variables, (const xmlChar *)
00265         variables_name))
00266         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00267             (xmlChar *) input->variables);
00268     file = g_file_new_for_path (input->directory);
00269     file2 = g_file_new_for_path (input->simulator);
00270     buffer = g_file_get_relative_path (file, file2);
00271     g_object_unref (file2);
00272     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00273     g_free (buffer);
00274     if (input->evaluator)
00275     {

```

```

00275     file2 = g_file_new_for_path (input->evaluator);
00276     buffer = g_file_get_relative_path (file, file2);
00277     g_object_unref (file2);
00278     if (xmlStrlen ((xmlChar *) buffer))
00279         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00280                     (xmlChar *) buffer);
00281     g_free (buffer);
00282 }
00283 if (input->seed != DEFAULT_RANDOM_SEED)
00284     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);

00285
00286 // Setting the algorithm
00287 buffer = (char *) g_slice_alloc (64);
00288 switch (input->algorithm)
00289 {
00290     case ALGORITHM_MONTE_CARLO:
00291         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00292                     (const xmlChar *) LABEL_MONTE_CARLO);
00293         snprintf (buffer, 64, "%u", input->nsimulations);
00294         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00295                     (xmlChar *) buffer);
00296         snprintf (buffer, 64, "%u", input->niterations);
00297         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00298                     (xmlChar *) buffer);
00299         snprintf (buffer, 64, "%.3lg", input->tolerance);
00300         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00301                     (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%u", input->nbest);
00303         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304         input_save_direction_xml (node);
00305         break;
00306     case ALGORITHM_SWEEP:
00307         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                     (const xmlChar *) LABEL_SWEEP);
00309         snprintf (buffer, 64, "%u", input->niterations);
00310         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                     (xmlChar *) buffer);
00312         snprintf (buffer, 64, "%.3lg", input->tolerance);
00313         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00314                     (xmlChar *) buffer);
00315         snprintf (buffer, 64, "%u", input->nbest);
00316         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00317         input_save_direction_xml (node);
00318         break;
00319     default:
00320         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00321                     (const xmlChar *) LABEL_GENETIC);
00322         snprintf (buffer, 64, "%u", input->nsimulations);
00323         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00324                     (xmlChar *) buffer);
00325         snprintf (buffer, 64, "%u", input->niterations);
00326         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00327                     (xmlChar *) buffer);
00328         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00329         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00332                     (xmlChar *) buffer);
00333         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION,
00335                     (xmlChar *) buffer);
00336         break;
00337 }
00338 g_slice_free1 (64, buffer);
00339 if (input->threshold != 0.)
00340     xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,
00341                         input->threshold);
00342
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 *)
LABEL_WEIGHT,
00351                             input->experiment[i].weight);
00352     for (j = 0; j < input->experiment->ninputs; ++j)
00353         xmlSetProp (child, (const xmlChar *) template[j],
00354                     (xmlChar *) input->experiment[i].template[j]);
00355 }
00356
00357 // Setting the variables data
00358 for (i = 0; i < input->nvariables; ++i)

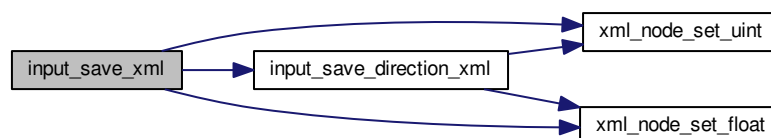
```

```

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             xml_node_set_uint (child, (const xmlChar *)
00384                 LABEL_NSWEEPS,
00385                 input->variable[i].nsweeps);
00386         else if (input->algorithm == ALGORITHM_GENETIC)
00387             xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00388                 input->variable[i].nbits);
00389         if (input->nsteps)
00390             xml_node_set_float (child, (const xmlChar *)
00391                 LABEL_STEP,
00392                 input->variable[i].step);
00393     }
00394 // Saving the error norm
00395 switch (input->norm)
00396 {
00397     case ERROR_NORM_MAXIMUM:
00398         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00399             (const xmlChar *) LABEL_MAXIMUM);
00400         break;
00401     case ERROR_NORM_P:
00402         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00403             (const xmlChar *) LABEL_P);
00404         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00405             input->p);
00406         break;
00407     case ERROR_NORM_TAXICAB:
00408         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00409             (const xmlChar *) LABEL_TAXICAB);
00410         break;
00411 }
00412 #if DEBUG_INTERFACE
00413 fprintf (stderr, "input_save: end\n");
00414 #endif
00415 }

```

Here is the call graph for this function:



4.11.2.6 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

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

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

Here is the call graph for this function:



4.11.2.7 window_get_direction()

```
unsigned int window_get_direction ( )
```

Function to get the direction search method number.

Returns

Direction search method number.

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

```

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

```

Here is the call graph for this function:



4.11.2.8 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

Returns

Norm method number.

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

```

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

```

Here is the call graph for this function:



4.11.2.9 window_new()

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

Function to open the main window.

Parameters

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

Definition at line 2108 of file [interface.c](#).

```
02109 {
02110     unsigned int i;
02111     char *buffer, *buffer2, buffer3[64];
02112     char *label_algorithm[NALGORITHMS] = {
02113         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114     };
02115     char *tip_algorithm[NALGORITHMS] = {
02116         _("Monte-Carlo brute force algorithm"),
02117         _("Sweep brute force algorithm"),
02118         _("Genetic algorithm")
02119     };
02120     char *label_direction[NDIRECTIONS] = {
02121         _("_Coordinates descent"), _("_Random")
02122     };
02123     char *tip_direction[NDIRECTIONS] = {
02124         _("Coordinates direction estimate method"),
02125         _("Random direction estimate method")
02126     };
02127     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128     char *tip_norm[NNORMS] = {
02129         _("Euclidean error norm (L2)"),
02130         _("Maximum error norm (L)"),
02131         _("P error norm (Lp)"),
02132         _("Taxicab error norm (L1)")
02133     };
02134
02135     #if DEBUG_INTERFACE
02136         fprintf (stderr, "window_new: start\n");
02137     #endif
02138
02139     // Creating the window
02140     window->window = main_window
02141         = (GtkWindow *) gtk_application_window_new (application);
02142
02143     // Finish when closing the window
02144     g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148     // Setting the window title
02149     gtk_window_set_title (window->window, "MPCOTool");
02150
02151     // Creating the open button
02152     window->button_open = (GtkToolButton *) gtk_tool_button_new
02153         (gtk_image_new_from_icon_name ("document-open",
02154                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157     // Creating the save button
02158     window->button_save = (GtkToolButton *) gtk_tool_button_new
02159         (gtk_image_new_from_icon_name ("document-save",
02160                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161     g_signal_connect (window->button_save, "clicked", (void (*)(
02162         window_save,
02163         NULL);
02164
02165     // Creating the run button
02166     window->button_run = (GtkToolButton *) gtk_tool_button_new
02167         (gtk_image_new_from_icon_name ("system-run",
02168                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02169     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02170
02171     // Creating the options button
02172     window->button_options = (GtkToolButton *) gtk_tool_button_new
```

```

02172     (gtk_image_new_from_icon_name ("preferences-system",
02173                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02174     _("Options"));
02175     g_signal_connect (window->button_options, "clicked",
02176                       options_new, NULL);
02177
02178     // Creating the help button
02179     window->button_help = (GtkToolButton *) gtk_tool_button_new
02180     (gtk_image_new_from_icon_name ("help-browser",
02181                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02182     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02183
02184     // Creating the about button
02185     window->button_about = (GtkToolButton *) gtk_tool_button_new
02186     (gtk_image_new_from_icon_name ("help-about",
02187                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02188     g_signal_connect (window->button_about, "clicked",
02189                       window_about, NULL);
02190
02191     // Creating the exit button
02192     window->button_exit = (GtkToolButton *) gtk_tool_button_new
02193     (gtk_image_new_from_icon_name ("application-exit",
02194                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02195     g_signal_connect_swapped (window->button_exit, "clicked",
02196                               G_CALLBACK (g_application_quit),
02197                               G_APPLICATION (application));
02198
02199     // Creating the buttons bar
02200     window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02201     gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->
02203     button_open), 0);
02204     gtk_toolbar_insert
02205     (window->bar_buttons, GTK_TOOL_ITEM (window->
02206     button_save), 1);
02207     gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->
02209     button_run), 2);
02210     gtk_toolbar_insert
02211     (window->bar_buttons, GTK_TOOL_ITEM (window->
02212     button_options), 3);
02213     gtk_toolbar_insert
02214     (window->bar_buttons, GTK_TOOL_ITEM (window->
02215     button_help), 4);
02216     gtk_toolbar_insert
02217     (window->bar_buttons, GTK_TOOL_ITEM (window->
02218     button_about), 5);
02219     gtk_toolbar_insert
02220     (window->bar_buttons, GTK_TOOL_ITEM (window->
02221     button_exit), 6);
02222     gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02223
02224     // Creating the simulator program label and entry
02225     window->label_simulator
02226     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02227     window->button_simulator = (GtkFileChooserButton *)
02228     gtk_file_chooser_button_new (_("Simulator program"),
02229                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02230     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02231                                  _("Simulator program executable file"));
02232     gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02233
02234     // Creating the evaluator program label and entry
02235     window->check_evaluator = (GtkCheckButton *)
02236     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02237     g_signal_connect (window->check_evaluator, "toggled",
02238                       window_update, NULL);
02239     window->button_evaluator = (GtkFileChooserButton *)
02240     gtk_file_chooser_button_new (_("Evaluator program"),
02241                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02242     gtk_widget_set_tooltip_text
02243     (GTK_WIDGET (window->button_evaluator),
02244      _("Optional evaluator program executable file"));
02245
02246     // Creating the results files labels and entries
02247     window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02248     window->entry_result = (GtkEntry *) gtk_entry_new ();
02249     gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->entry_result), _("Best results file"));
02251     window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02252     window->entry_variables = (GtkEntry *) gtk_entry_new ();
02253     gtk_widget_set_tooltip_text
02254     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02255
02256     // Creating the files grid and attaching widgets
02257     window->grid_files = (GtkGrid *) gtk_grid_new ();
02258     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->

```

```

    label_simulator),
02249         0, 0, 1, 1);
02250     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02251         1, 0, 1, 1);
02252     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02253         0, 1, 1, 1);
02254     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02255         1, 1, 1, 1);
02256     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02257         0, 2, 1, 1);
02258     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02259         1, 2, 1, 1);
02260     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02261         0, 3, 1, 1);
02262     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02263         1, 3, 1, 1);
02264
02265     // Creating the algorithm properties
02266     window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268     window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270     gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272     _("Number of simulations to perform for each iteration"));
02273     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274     window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276     window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278     gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280     g_signal_connect
02281     (window->spin_iterations, "value-changed",
window_update, NULL);
02282     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284     window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286     gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288     _("Tolerance to set the variable interval on the next iteration"));
02289     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290     window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292     gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294     _("Number of best simulations used to set the variable interval "
02295     "on the next iteration"));
02296     window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298     window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300     gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302     _("Number of population for the genetic algorithm"));
02303     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304     window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306     window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308     gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310     _("Number of generations for the genetic algorithm"));
02311     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312     window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314     gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316     _("Ratio of mutation for the genetic algorithm"));
02317     window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319     window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321     gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323     _("Ratio of reproduction for the genetic algorithm"));
02324     window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326     window->spin_adaptation

```

```

02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328     gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332     window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335     gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338     window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                        GTK_WIDGET (window->spin_threshold));
02342     // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343     // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344     //                          GTK_ALIGN_FILL);
02345
02346     // Creating the direction search method properties
02347     window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02349     g_signal_connect (window->check_direction, "clicked",
02350                      window_update, NULL);
02351     window->grid_direction = (GtkGrid *) gtk_grid_new ();
02352     window->button_direction[0] = (GtkRadioButton *)
02353     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02354     gtk_grid_attach (window->grid_direction,
02355                     GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02356     g_signal_connect (window->button_direction[0], "clicked",
02357                      window_update, NULL);
02358     for (i = 0; ++i < NDIRECTIONS;)
02359     {
02360         window->button_direction[i] = (GtkRadioButton *)
02361         gtk_radio_button_new_with_mnemonic
02362         (gtk_radio_button_get_group (window->button_direction[0]),
02363          label_direction[i]);
02364         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02365                                     tip_direction[i]);
02366         gtk_grid_attach (window->grid_direction,
02367                         GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02368         g_signal_connect (window->button_direction[i], "clicked",
02369                          window_update, NULL);
02370     }
02371     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02372     window->spin_steps = (GtkSpinButton *)
02373     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02374     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02375     window->label_estimates
02376     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02377     window->spin_estimates = (GtkSpinButton *)
02378     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02379     window->label_relaxation
02380     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02381     window->spin_relaxation = (GtkSpinButton *)
02382     gtk_spin_button_new_with_range (0., 2., 0.001);
02383     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02384     window->label_steps),
02385                     0, NDIRECTIONS, 1, 1);
02386     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02387     window->spin_steps),
02388                     1, NDIRECTIONS, 1, 1);
02389     gtk_grid_attach (window->grid_direction,
02390                     GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02391                     1, 1);
02392     gtk_grid_attach (window->grid_direction,
02393                     GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02394                     1);
02395     gtk_grid_attach (window->grid_direction,
02396                     GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02397                     1, 1);
02398     gtk_grid_attach (window->grid_direction,
02399                     GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02400                     1, 1);
02401
02402     // Creating the array of algorithms
02403     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02404     window->button_algorithm[0] = (GtkRadioButton *)
02405     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02406     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02407                                 tip_algorithm[0]);
02408     gtk_grid_attach (window->grid_algorithm,
02409                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02410     g_signal_connect (window->button_algorithm[0], "clicked",
02411                      window_set_algorithm, NULL);
02412     for (i = 0; ++i < NALGORITHMS;)

```

```

02410     {
02411         window->button_algorithm[i] = (GtkRadioButton *)
02412             gtk_radio_button_new_with_mnemonic
02413             (gtk_radio_button_get_group (window->button_algorithm[0]),
02414              label_algorithm[i]);
02415         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416                                     tip_algorithm[i]);
02417         gtk_grid_attach (window->grid_algorithm,
02418                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419         g_signal_connect (window->button_algorithm[i], "clicked",
02420                          window_set_algorithm, NULL);
02421     }
02422     gtk_grid_attach (window->grid_algorithm,
02423                     GTK_WIDGET (window->label_simulations), 0,
02424                     NALGORITHMS, 1, 1);
02425     gtk_grid_attach (window->grid_algorithm,
02426                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427                     1);
02428     gtk_grid_attach (window->grid_algorithm,
02429                     GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430                     1, 1);
02431     gtk_grid_attach (window->grid_algorithm,
02432                     GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433                     1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435                     GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436                     1, 1);
02437     gtk_grid_attach (window->grid_algorithm,
02438                     GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439                     1);
02440     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02441 window->label_bests),
02442                     0, NALGORITHMS + 3, 1, 1);
02443     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02444 window->spin_bests), 1,
02445                     NALGORITHMS + 3, 1, 1);
02446     gtk_grid_attach (window->grid_algorithm,
02447                     GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02448                     1, 1);
02449     gtk_grid_attach (window->grid_algorithm,
02450                     GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02451                     1, 1);
02452     gtk_grid_attach (window->grid_algorithm,
02453                     GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02454                     1, 1);
02455     gtk_grid_attach (window->grid_algorithm,
02456                     GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02457                     1, 1);
02458     gtk_grid_attach (window->grid_algorithm,
02459                     GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02460                     1);
02461     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02462 window->spin_mutation),
02463                     1, NALGORITHMS + 6, 1, 1);
02464     gtk_grid_attach (window->grid_algorithm,
02465                     GTK_WIDGET (window->label_reproduction), 0,
02466                     NALGORITHMS + 7, 1, 1);
02467     gtk_grid_attach (window->grid_algorithm,
02468                     GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02469                     1, 1);
02470     gtk_grid_attach (window->grid_algorithm,
02471                     GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02472                     1, 1);
02473     gtk_grid_attach (window->grid_algorithm,
02474                     GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02475                     1, 1);
02476     gtk_grid_attach (window->grid_algorithm,
02477                     GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02478                     2, 1);
02479     gtk_grid_attach (window->grid_algorithm,
02480                     GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02481                     2, 1);
02482     gtk_grid_attach (window->grid_algorithm,
02483                     GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02484                     1, 1);
02485     gtk_grid_attach (window->grid_algorithm,
02486                     GTK_WIDGET (window->scrolled_threshold), 1,
02487                     NALGORITHMS + 11, 1, 1);
02488     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02489     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02490                       GTK_WIDGET (window->grid_algorithm));
02491     // Creating the variable widgets
02492     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02493     gtk_widget_set_tooltip_text
02494         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02495     window->id_variable = g_signal_connect

```

```

02494     (window->combo_variable, "changed", window_set_variable, NULL);
02495     window->button_add_variable
02496     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497                                                    GTK_ICON_SIZE_BUTTON);
02498     g_signal_connect
02499     (window->button_add_variable, "clicked",
02500      window_add_variable, NULL);
02501     gtk_widget_set_tooltip_text
02502     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02503     window->button_remove_variable
02504     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02505                                                    GTK_ICON_SIZE_BUTTON);
02506     g_signal_connect
02507     (window->button_remove_variable, "clicked",
02508      window_remove_variable, NULL);
02509     gtk_widget_set_tooltip_text
02510     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02511     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02512     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02513     gtk_widget_set_tooltip_text
02514     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02515     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02516     window->id_variable_label = g_signal_connect
02517     (window->entry_variable, "changed",
02518      window_label_variable, NULL);
02519     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02520     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02521     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02522     gtk_widget_set_tooltip_text
02523     (GTK_WIDGET (window->spin_min),
02524      _("Minimum initial value of the variable"));
02525     window->scrolled_min
02526     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02527     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02528                        GTK_WIDGET (window->spin_min));
02529     g_signal_connect (window->spin_min, "value-changed",
02530                      window_rangemin_variable, NULL);
02531     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02532     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02533     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02534     gtk_widget_set_tooltip_text
02535     (GTK_WIDGET (window->spin_max),
02536      _("Maximum initial value of the variable"));
02537     window->scrolled_max
02538     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02539     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02540                        GTK_WIDGET (window->spin_max));
02541     g_signal_connect (window->spin_max, "value-changed",
02542                      window_rangemax_variable, NULL);
02543     window->check_minabs = (GtkCheckButton *)
02544     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02545     g_signal_connect (window->check_minabs, "toggled",
02546                      window_update, NULL);
02547     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02548     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02549     gtk_widget_set_tooltip_text
02550     (GTK_WIDGET (window->spin_minabs),
02551      _("Minimum allowed value of the variable"));
02552     window->scrolled_minabs
02553     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02554     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02555                        GTK_WIDGET (window->spin_minabs));
02556     g_signal_connect (window->spin_minabs, "value-changed",
02557                      window_rangeminabs_variable, NULL);
02558     window->check_maxabs = (GtkCheckButton *)
02559     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02560     g_signal_connect (window->check_maxabs, "toggled",
02561                      window_update, NULL);
02562     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02563     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02564     gtk_widget_set_tooltip_text
02565     (GTK_WIDGET (window->spin_maxabs),
02566      _("Maximum allowed value of the variable"));
02567     window->scrolled_maxabs
02568     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02569     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02570                        GTK_WIDGET (window->spin_maxabs));
02571     g_signal_connect (window->spin_maxabs, "value-changed",
02572                      window_rangemaxabs_variable, NULL);
02573     window->label_precision
02574     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02575     window->spin_precision = (GtkSpinButton *)
02576     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02577     gtk_widget_set_tooltip_text
02578     (GTK_WIDGET (window->spin_precision),
02579      _("Number of precision floating point digits\n"
02580        "0 is for integer numbers"));

```



```

02576 g_signal_connect (window->spin_precision, "value-changed",
02577                  window_precision_variable, NULL);
02578 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579 window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582                             _("Number of steps sweeping the variable"));
02583 g_signal_connect (window->spin_sweeps, "value-changed",
02584                  window_update_variable, NULL);
02585 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586 window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588 gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590      _("Number of bits to encode the variable"));
02591 g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL)
02593 ;
02594 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02595 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02596     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02597 gtk_widget_set_tooltip_text
02598     (GTK_WIDGET (window->spin_step),
02599      _("Initial step size for the direction search method"));
02599 window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602                  GTK_WIDGET (window->spin_step));
02603 g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606 gtk_grid_attach (window->grid_variable,
02607                 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608 gtk_grid_attach (window->grid_variable,
02609                 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610 gtk_grid_attach (window->grid_variable,
02611                 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613                 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614 gtk_grid_attach (window->grid_variable,
02615                 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616 gtk_grid_attach (window->grid_variable,
02617                 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618 gtk_grid_attach (window->grid_variable,
02619                 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620 gtk_grid_attach (window->grid_variable,
02621                 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622 gtk_grid_attach (window->grid_variable,
02623                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624 gtk_grid_attach (window->grid_variable,
02625                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626 gtk_grid_attach (window->grid_variable,
02627                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628 gtk_grid_attach (window->grid_variable,
02629                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630 gtk_grid_attach (window->grid_variable,
02631                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632 gtk_grid_attach (window->grid_variable,
02633                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634 gtk_grid_attach (window->grid_variable,
02635                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636 gtk_grid_attach (window->grid_variable,
02637                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638 gtk_grid_attach (window->grid_variable,
02639                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640 gtk_grid_attach (window->grid_variable,
02641                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642 gtk_grid_attach (window->grid_variable,
02643                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644 gtk_grid_attach (window->grid_variable,
02645                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646 gtk_grid_attach (window->grid_variable,
02647                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                  GTK_WIDGET (window->grid_variable));
02651
02652 // Creating the experiment widgets
02653 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                             _("Experiment selector"));
02656 window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed",
02658      window_set_experiment, NULL);
02659 window->button_add_experiment
02660     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02661                                                    GTK_ICON_SIZE_BUTTON);

```

```

02661 g_signal_connect
02662 (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02663 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664 _("Add experiment"));
02665 window->button_remove_experiment
02666 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667 GTK_ICON_SIZE_BUTTON);
02668 g_signal_connect (window->button_remove_experiment, "clicked",
02669 window_remove_experiment, NULL);
02670 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
button_remove_experiment),
02671 _("Remove experiment"));
02672 window->label_experiment
02673 = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674 window->button_experiment = (GtkFileChooserButton *)
02675 gtk_file_chooser_button_new (_("Experimental data file"),
02676 GTK_FILE_CHOOSER_ACTION_OPEN);
02677 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678 _("Experimental data file"));
02679 window->id_experiment_name
02680 = g_signal_connect (window->button_experiment, "selection-changed",
02681 window_name_experiment, NULL);
02682 gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02683 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684 window->spin_weight
02685 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02686 gtk_widget_set_tooltip_text
02687 (GTK_WIDGET (window->spin_weight),
02688 _("Weight factor to build the objective function"));
02689 g_signal_connect
02690 (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
02691 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692 gtk_grid_attach (window->grid_experiment,
02693 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694 gtk_grid_attach (window->grid_experiment,
02695 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696 gtk_grid_attach (window->grid_experiment,
02697 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
;
02698 gtk_grid_attach (window->grid_experiment,
02699 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700 gtk_grid_attach (window->grid_experiment,
02701 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702 gtk_grid_attach (window->grid_experiment,
02703 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704 gtk_grid_attach (window->grid_experiment,
02705 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706 for (i = 0; i < MAX_NINPUS; ++i)
02707 {
02708 snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709 window->check_template[i] = (GtkCheckButton *)
02710 gtk_check_button_new_with_label (buffer3);
02711 window->id_template[i]
02712 = g_signal_connect (window->check_template[i], "toggled",
02713 window_inputs_experiment, NULL);
02714 gtk_grid_attach (window->grid_experiment,
02715 GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716 1);
02717 window->button_template[i] =
02718 (GtkFileChooserButton *)
02719 gtk_file_chooser_button_new (_("Input template"),
02720 GTK_FILE_CHOOSER_ACTION_OPEN);
02721 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722 _("Experimental input template file"));
02723 window->id_input[i] =
02724 g_signal_connect_swapped (window->button_template[i],
02725 "selection-changed",
02726 (void (*)(void *)) window_template_experiment,
02727 (void *) (size_t) i);
02728 gtk_grid_attach (window->grid_experiment,
02729 GTK_WIDGET (window->button_template[i]),
02730 1, 3 + i, 3, 1);
02731 }
02732 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734 GTK_WIDGET (window->grid_experiment));
02735
02736 // Creating the error norm widgets
02737 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740 GTK_WIDGET (window->grid_norm));
02741 window->button_norm[0] = (GtkRadioButton *)
02742 gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),

```

```

02744         tip_norm[0]);
02745     gtk_grid_attach (window->grid_norm,
02746                     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747     g_signal_connect (window->button_norm[0], "clicked",
02748 window_update, NULL);
02749     for (i = 0; ++i < NNORMS;)
02750     {
02751         window->button_norm[i] = (GtkRadioButton *)
02752             gtk_radio_button_new_with_mnemonic
02753             (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02754         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02755             tip_norm[i]);
02756         gtk_grid_attach (window->grid_norm,
02757             GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02758         g_signal_connect (window->button_norm[i], "clicked",
02759 window_update,
02760             NULL);
02761     }
02762     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02763     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02764 label_p), 1, 1, 1,
02765     1);
02766     window->spin_p =
02767         (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02768             G_MAXDOUBLE, 0.01);
02769     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02770         _("P parameter for the P error norm"));
02771     window->scrolled_p =
02772         (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02773     gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02774         GTK_WIDGET (window->spin_p));
02775     gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02776     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02777     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02778 scrolled_p),
02779     1, 2, 1, 2);
02780 // Creating the grid and attaching the widgets to the grid
02781 window->grid = (GtkGrid *) gtk_grid_new ();
02782 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02783     1);
02784 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02785 gtk_grid_attach (window->grid,
02786     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02787 gtk_grid_attach (window->grid,
02788     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02789 gtk_grid_attach (window->grid,
02790     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02791 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02792 gtk_container_add (GTK_CONTAINER (window->window),
02793     GTK_WIDGET (window->grid));
02794 // Setting the window logo
02795 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02796 gtk_window_set_icon (window->window, window->logo);
02797 // Showing the window
02798 gtk_widget_show_all (GTK_WIDGET (window->window));
02799 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02800 #if GTK_MINOR_VERSION >= 16
02801     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02802     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02803     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02804     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02805     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02806     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02807     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02808         40);
02809 #endif
02810 // Reading initial example
02811 input_new ();
02812 buffer2 = g_get_current_dir ();
02813 buffer =
02814     g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02815 g_free (buffer2);
02816 window_read (buffer);
02817 g_free (buffer);
02818 #if DEBUG_INTERFACE
02819     fprintf (stderr, "window_new: start\n");
02820 #endif
02821 }

```

4.11.2.10 window_read()

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

Function to read the input data of a file.

Parameters

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

Returns

1 on succes, 0 on error.

Definition at line 1904 of file [interface.c](#).

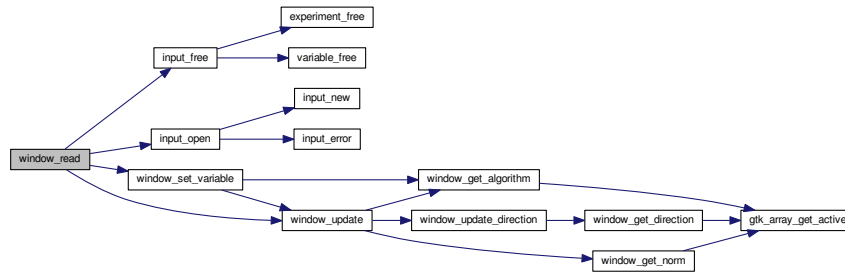
```
01905 {
01906     unsigned int i;
01907     char *buffer;
01908     #if DEBUG_INTERFACE
01909     fprintf (stderr, "window_read: start\n");
01910     #endif
01911
01912     // Reading new input file
01913     input_free ();
01914     if (!input_open (filename))
01915     {
01916         #if DEBUG_INTERFACE
01917         fprintf (stderr, "window_read: end\n");
01918         #endif
01919         return 0;
01920     }
01921
01922     // Setting GTK+ widgets data
01923     gtk_entry_set_text (window->entry_result, input->result);
01924     gtk_entry_set_text (window->entry_variables, input->
variables);
01925     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01926     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01927     g_free (buffer);
01928     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01929     if (input->evaluator)
01930     {
01931         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01932         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01933         g_free (buffer);
01934     }
01935     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01936     switch (input->algorithm)
01937     {
01938     case ALGORITHM_MONTE_CARLO:
01939         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01940     case ALGORITHM_SWEEP:
01941         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01942         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01943         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01944         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_direction),
input->nsteps);
01945         if (input->nsteps)
01946         {
01947             gtk_toggle_button_set_active
```

```

01955         (GTK_TOGGLE_BUTTON (window->button_direction
01956                             [input->direction]), TRUE);
01957     gtk_spin_button_set_value (window->spin_steps,
01958                              (gdouble) input->nsteps);
01959     gtk_spin_button_set_value (window->spin_relaxation,
01960                              (gdouble) input->relaxation);
01961     switch (input->direction)
01962     {
01963     case DIRECTION_METHOD_RANDOM:
01964         gtk_spin_button_set_value (window->spin_estimates,
01965                                  (gdouble) input->nestimates);
01966     }
01967     }
01968     break;
01969 default:
01970     gtk_spin_button_set_value (window->spin_population,
01971                              (gdouble) input->nsimulations);
01972     gtk_spin_button_set_value (window->spin_generations,
01973                              (gdouble) input->niterations);
01974     gtk_spin_button_set_value (window->spin_mutation,
01975                              input->mutation_ratio);
01976     gtk_spin_button_set_value (window->spin_reproduction,
01977                              input->reproduction_ratio);
01978     gtk_spin_button_set_value (window->spin_adaptation,
01979                              input->adaptation_ratio);
01980     }
01981     gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983     gtk_spin_button_set_value (window->spin_p, input->p);
01984     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01985     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01986     g_signal_handler_block (window->button_experiment,
01987                             window->id_experiment_name);
01988     gtk_combo_box_text_remove_all (window->combo_experiment);
01989     for (i = 0; i < input->nexperiments; ++i)
01990         gtk_combo_box_text_append_text (window->combo_experiment,
01991                                         input->experiment[i].name);
01992     g_signal_handler_unblock
01993     (window->button_experiment, window->
id_experiment_name);
01994     g_signal_handler_unblock (window->combo_experiment,
01995                             window->id_experiment);
01996     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01997     g_signal_handler_block (window->combo_variable, window->
id_variable);
01998     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01999     gtk_combo_box_text_remove_all (window->combo_variable);
02000     for (i = 0; i < input->nvariables; ++i)
02001         gtk_combo_box_text_append_text (window->combo_variable,
02002                                         input->variable[i].name);
02003     g_signal_handler_unblock (window->entry_variable,
02004                             window->id_variable_label);
02005     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
02006     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02007     window_set_variable ();
02008     window_update ();
02009 #if DEBUG_INTERFACE
02010     fprintf (stderr, "window_read: end\n");
02011 #endif
02012     return 1;
02013 }

```

Here is the call graph for this function:



4.11.2.11 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 825 of file [interface.c](#).

```

00826 {
00827     GtkFileChooserDialog *dlg;
00828     GtkFileFilter *filter1, *filter2;
00829     char *buffer;
00830
00831     #if DEBUG_INTERFACE
00832     fprintf (stderr, "window_save: start\n");
00833     #endif
00834
00835     // Opening the saving dialog
00836     dlg = (GtkFileChooserDialog *)
00837         gtk_file_chooser_dialog_new (_, "Save file",
00838                                     window->window,
00839                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                     _("_Cancel"),
00841                                     GTK_RESPONSE_CANCEL,
00842                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00843     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844                                                    TRUE);
00845     buffer = g_build_filename (input->directory, input->name, NULL);
00846     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847     g_free (buffer);
00848
00849     // Adding XML filter
00850     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter1, "XML");
00852     gtk_file_filter_add_pattern (filter1, "*.xml");
00853     gtk_file_filter_add_pattern (filter1, "*.XML");
00854     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856     // Adding JSON filter
00857     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858     gtk_file_filter_set_name (filter2, "JSON");
00859     gtk_file_filter_add_pattern (filter2, "*.json");
00860     gtk_file_filter_add_pattern (filter2, "*.JSON");
00861     gtk_file_filter_add_pattern (filter2, "*.js");
00862     gtk_file_filter_add_pattern (filter2, "*.JS");
00863     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);

```

```

00864
00865     if (input->type == INPUT_TYPE_XML)
00866         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867     else
00868         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870     // If OK response then saving
00871     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872     {
00873         // Setting input file type
00874         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875         buffer = (char *) gtk_file_filter_get_name (filter1);
00876         if (!strcmp (buffer, "XML"))
00877             input->type = INPUT_TYPE_XML;
00878         else
00879             input->type = INPUT_TYPE_JSON;
00880
00881         // Adding properties to the root XML node
00882         input->simulator = gtk_file_chooser_get_filename
00883             (GTK_FILE_CHOOSER (window->button_simulator));
00884         if (gtk_toggle_button_get_active
00885             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886             input->evaluator = gtk_file_chooser_get_filename
00887                 (GTK_FILE_CHOOSER (window->button_evaluator));
00888         else
00889             input->evaluator = NULL;
00890         if (input->type == INPUT_TYPE_XML)
00891         {
00892             input->result
00893                 = (char *) xmlStrdup ((const xmlChar *)
00894                                         gtk_entry_get_text (window->entry_result));
00895             input->variables
00896                 = (char *) xmlStrdup ((const xmlChar *)
00897                                         gtk_entry_get_text
00898                                             (window->entry_variables));
00899         }
00900         else
00901         {
00902             input->result =
00903                 g_strdup (gtk_entry_get_text (window->entry_result));
00904             input->variables =
00905                 g_strdup (gtk_entry_get_text (window->entry_variables));
00906         }
00907
00908         // Setting the algorithm
00909         switch (window_get_algorithm ())
00910         {
00911             case ALGORITHM_MONTE_CARLO:
00912                 input->algorithm = ALGORITHM_MONTE_CARLO;
00913                 input->nsimulations
00914                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915                 input->niterations
00916                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917                 input->tolerance =
00918                     gtk_spin_button_get_value (window->spin_tolerance);
00919                 input->nbest =
00920                     gtk_spin_button_get_value_as_int (window->spin_bests);
00921                 window_save_direction ();
00922                 break;
00923             case ALGORITHM_SWEEP:
00924                 input->algorithm = ALGORITHM_SWEEP;
00925                 input->niterations
00926                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927                 input->tolerance =
00928                     gtk_spin_button_get_value (window->spin_tolerance);
00929                 input->nbest =
00930                     gtk_spin_button_get_value_as_int (window->spin_bests);
00931                 window_save_direction ();
00932                 break;
00933             default:
00934                 input->algorithm = ALGORITHM_GENETIC;
00935                 input->nsimulations
00936                     = gtk_spin_button_get_value_as_int (window->spin_population);
00937                 input->niterations
00938                     = gtk_spin_button_get_value_as_int (window->spin_generations);
00939                 input->mutation_ratio
00940                     = gtk_spin_button_get_value (window->spin_mutation);
00941                 input->reproduction_ratio
00942                     = gtk_spin_button_get_value (window->spin_reproduction);
00943                 input->adaptation_ratio
00944                     = gtk_spin_button_get_value (window->spin_adaptation);
00945                 break;
00946         }
00947         input->norm = window_get_norm ();
00948         input->p = gtk_spin_button_get_value (window->spin_p);
00949         input->threshold = gtk_spin_button_get_value (window->
spin_threshold);

```

```

00950
00951     // Saving the XML file
00952     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953     input_save (buffer);
00954
00955     // Closing and freeing memory
00956     g_free (buffer);
00957     gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959     fprintf (stderr, "window_save: end\n");
00960 #endif
00961     return 1;
00962 }
00963
00964 // Closing and freeing memory
00965 gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967     fprintf (stderr, "window_save: end\n");
00968 #endif
00969     return 0;
00970 }

```

4.11.2.12 window_template_experiment()

```

void window_template_experiment (
    void * data )

```

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

Parameters

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

Definition at line 1542 of file [interface.c](#).

```

01543 {
01544     unsigned int i, j;
01545     char *buffer;
01546     GFile *file1, *file2;
01547 #if DEBUG_INTERFACE
01548     fprintf (stderr, "window_template_experiment: start\n");
01549 #endif
01550     i = (size_t) data;
01551     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552     file1
01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555     file2 = g_file_new_for_path (input->directory);
01556     buffer = g_file_get_relative_path (file2, file1);
01557     if (input->type == INPUT_TYPE_XML)
01558         input->experiment[j].template[i] =
01559         (char *) xmlStrdup ((xmlChar *) buffer);
01560     else
01561         input->experiment[j].template[i] = g_strdup (buffer);
01562     g_free (buffer);
01563     g_object_unref (file2);
01564     g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566     fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }

```

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 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #include <gio/gio.h>
00051 #include <gtk/gtk.h>
00052 #include "genetic/genetic.h"
00053 #include "utils.h"
00054 #include "experiment.h"
00055 #include "variable.h"
00056 #include "input.h"
00057 #include "optimize.h"
00058 #include "interface.h"
00059
00060 #define DEBUG_INTERFACE 0
00061
00062 #ifdef G_OS_WIN32
00063 #define INPUT_FILE "test-ga-win.xml"
00064 #else
00065 #define INPUT_FILE "test-ga.xml"
00066 #endif
00067
00068 const char *logo[] = {
00069     "32 32 3 1",
00070     "    c None",
00071     ".    c #0000FF",
00072     "+    c #FF0000",
00073     " ",
00074     " ",
00075     " ",
00076     " . . . .",
00077     " . . . .",
00078     " . . . .",
00079     " . . . .",
00080     " . . . .",
00081     " . . . .",
00082     " . . . .",
00083     " . . . .",
00084     " . . . .",
00085     " . . . .",
00086     " . . . .",
00087     " . . . .",
00088     " . . . .",
00089     " . . . .",
00090     " . . . .",
00091     " . . . .",
00092     " . . . .",
00093     " . . . .",
00094     " . . . .",
00095     " . . . .",
00096     " . . . .",
00097     " . . . .",
00098     " . . . .",
00099     " . . . .",
00100     " . . . .",
00101     " . . . .",
00102     " . . . .",
00103     " . . . .",
00104     " . . . .",
00105     " . . . .",
00106     " . . . .",
00107     " . . . .",
00108     " . . . .",
00109     " . . . .",
00110     " . . . .",
00111     " . . . .",
00112     " . . . .",
00113     " . . . .",
00114     " . . . .",
00115     " . . . .",
00116     " . . . .",
00117     " . . . .",
00118     " . . . .",
00119     " . . . .",
00120     " . . . .",
00121     " . . . .",
00122     " . . . .",
00123     " . . . .",
00124     " . . . .",
00125     " . . . .",
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     " . . . .",
00161     " . . . .",
00162     " . . . .",
00163     " . . . .",
00164     " . . . .",
00165     " . . . .",
00166     " . . . .",
00167     " . . . .",
00168     " . . . .",
00169     " . . . .",
00170     " . . . .",
00171     " . . . .",
00172     " . . . .",
00173     " . . . .",
00174     " . . . .",
00175     " . . . .",
00176     " . . . .",
00177     " . . . .",
00178     " . . . .",
00179     " . . . .",
00180     " . . . .",
00181     " . . . .",
00182     " . . . .",
00183     " . . . .",
00184     " . . . .",
00185     " . . . .",
00186     " . . . .",
00187     " . . . .",
00188     " . . . .",
00189     " . . . .",
00190     " . . . .",
00191     " . . . .",
00192     " . . . .",
00193     " . . . .",
00194     " . . . .",
00195     " . . . .",
00196     " . . . .",
00197     " . . . .",
00198     " . . . .",
00199     " . . . .",
00200     " . . . .",
00201     " . . . .",
00202     " . . . .",
00203     " . . . .",
00204     " . . . .",
00205     " . . . .",
00206     " . . . .",
00207     " . . . .",
00208     " . . . .",
00209     " . . . .",
00210     " . . . .",
00211     " . . . .",
00212     " . . . .",
00213     " . . . .",
00214     " . . . .",
00215     " . . . .",
00216     " . . . .",
00217     " . . . .",
00218     " . . . .",
00219     " . . . .",
00220     " . . . .",
00221     " . . . .",
00222     " . . . .",
00223     " . . . .",
00224     " . . . .",
00225     " . . . .",
00226     " . . . .",
00227     " . . . .",
00228     " . . . .",
00229     " . . . .",
00230     " . . . .",
00231     " . . . .",
00232     " . . . .",
00233     " . . . .",
00234     " . . . .",
00235     " . . . .",
00236     " . . . .",
00237     " . . . .",
00238     " . . . .",
00239     " . . . .",
00240     " . . . .",
00241     " . . . .",
00242     " . . . .",
00243     " . . . .",
00244     " . . . .",
00245     " . . . .",
00246     " . . . .",
00247     " . . . .",
00248     " . . . .",
00249     " . . . .",
00250     " . . . .",
00251     " . . . .",
00252     " . . . .",
00253     " . . . .",
00254     " . . . .",
00255     " . . . .",
00256     " . . . .",
00257     " . . . .",
00258     " . . . .",
00259     " . . . .",
00260     " . . . .",
00261     " . . . .",
00262     " . . . .",
00263     " . . . .",
00264     " . . . .",
00265     " . . . .",
00266     " . . . .",
00267     " . . . .",
00268     " . . . .",
00269     " . . . .",
00270     " . . . .",
00271     " . . . .",
00272     " . . . .",
00273     " . . . .",
00274     " . . . .",
00275     " . . . .",
00276     " . . . .",
00277     " . . . .",
00278     " . . . .",
00279     " . . . .",
00280     " . . . .",
00281     " . . . .",
00282     " . . . .",
00283     " . . . .",
00284     " . . . .",
00285     " . . . .",
00286     " . . . .",
00287     " . . . .",
00288     " . . . .",
00289     " . . . .",
00290     " . . . .",
00291     " . . . .",
00292     " . . . .",
00293     " . . . .",
00294     " . . . .",
00295     " . . . .",
00296     " . . . .",
00297     " . . . .",
00298     " . . . .",
00299     " . . . .",
00300     " . . . .",
00301     " . . . .",
00302     " . . . .",
00303     " . . . .",
00304     " . . . .",
00305     " . . . .",
00306     " . . . .",
00307     " . . . .",
00308     " . . . .",
00309     " . . . .",
00310     " . . . .",
00311     " . . . .",
00312     " . . . .",
00313     " . . . .",
00314     " . . . .",
00315     " . . . .",
00316     " . . . .",
00317     " . . . .",
00318     " . . . .",
00319     " . . . .",
00320     " . . . .",
00321     " . . . .",
00322     " . . . .",
00323     " . . . .",
00324     " . . . .",
00325     " . . . .",
00326     " . . . .",
00327     " . . . .",
00328     " . . . .",
00329     " . . . .",
00330     " . . . .",
00331     " . . . .",
00332     " . . . .",
00333     " . . . .",
00334     " . . . .",
00335     " . . . .",
00336     " . . . .",
00337     " . . . .",
00338     " . . . .",
00339     " . . . .",
00340     " . . . .",
00341     " . . . .",
00342     " . . . .",
00343     " . . . .",
00344     " . . . .",
00345     " . . . .",
00346     " . . . .",
00347     " . . . .",
00348     " . . . .",
00349     " . . . .",
00350     " . . . .",
00351     " . . . .",
00352     " . . . .",
00353     " . . . .",
00354     " . . . .",
00355     " . . . .",
00356     " . . . .",
00357     " . . . .",
00358     " . . . .",
00359     " . . . .",
00360     " . . . .",
00361     " . . . .",
00362     " . . . .",
00363     " . . . .",
00364     " . . . .",
00365     " . . . .",
00366     " . . . .",
00367     " . . . .",
00368     " . . . .",
00369     " . . . .",
00370     " . . . .",
00371     " . . . .",
00372     " . . . .",
00373     " . . . .",
00374     " . . . .",
00375     " . . . .",
00376     " . . . .",
00377     " . . . .",
00378     " . . . .",
00379     " . . . .",
00380     " . . . .",
00381     " . . . .",
00382     " . . . .",
00383     " . . . .",
00384     " . . . .",
00385     " . . . .",
00386     " . . . .",
00387     " . . . .",
00388     " . . . .",
00389     " . . . .",
00390     " . . . .",
00391     " . . . .",
00392     " . . . .",
00393     " . . . .",
00394     " . . . .",
00395     " . . . .",
00396     " . . . .",
00397     " . . . .",
00398     " . . . .",
00399     " . . . .",
00400     " . . . .",
00401     " . . . .",
00402     " . . . .",
00403     " . . . .",
00404     " . . . .",
00405     " . . . .",
00406     " . . . .",
00407     " . . . .",
00408     " . . . .",
00409     " . . . .",
00410     " . . . .",
00411     " . . . .",
00412     " . . . .",
00413     " . . . .",
00414     " . . . .",
00415     " . . . .",
00416     " . . . .",
00417     " . . . .",
00418     " . . . .",
00419     " . . . .",
00420     " . . . .",
00421     " . . . .",
00422     " . . . .",
00423     " . . . .",
00424     " . . . .",
00425     " . . . .",
00426     " . . . .",
00427     " . . . .",
00428     " . . . .",
00429     " . . . .",
00430     " . . . .",
00431     " . . . .",
00432     " . . . .",
00433     " . . . .",
00434     " . . . .",
00435     " . . . .",
00436     " . . . .",
00437     " . . . .",
00438     " . . . .",
00439     " . . . .",
00440     " . . . .",
00441     " . . . .",
00442     " . . . .",
00443     " . . . .",
00444     " . . . .",
00445     " . . . .",
00446     " . . . .",
00447     " . . . .",
00448     " . . . .",
00449     " . . . .",
00450     " . . . .",
00451     " . . . .",
00452     " . . . .",
00453     " . . . .",
00454     " . . . .",
00455     " . . . .",
00456     " . . . .",
00457     " . . . .",
00458     " . . . .",
00459     " . . . .",
00460     " . . . .",
00461     " . . . .",
00462     " . . . .",
00463     " . . . .",
00464     " . . . .",
00465     " . . . .",
00466     " . . . .",
00467     " . . . .",
00468     " . . . .",
00469     " . . . .",
00470     " . . . .",
00471     " . . . .",
00472     " . . . .",
00473     " . . . .",
00474     " . . . .",
00475     " . . . .",
00476     " . . . .",
00477     " . . . .",
00478     " . . . .",
00479     " . . . .",
00480     " . . . .",
00481     " . . . .",
00482     " . . . .",
00483     " . . . .",
00484     " . . . .",
00485     " . . . .",
00486     " . . . .",
00487     " . . . .",
00488     " . . . .",
00489     " . . . .",
00490     " . . . .",
00491     " . . . .",
00492     " . . . .",
00493     " . . . .",
00494     " . . . .",
00495     " . . . .",
00496     " . . . .",
00497     " . . . .",
00498     " . . . .",
00499     " . . . .",
00500     " . . . .",
00501     " . . . .",
00502     " . . . .",
00503     " . . . .",
00504     " . . . .",
00505     " . . . .",
00506     " . . . .",
00507     " . . . .",
00508     " . . . .",
00509     " . . . .",
00510     " . . . .",
00511     " . . . .",
00512     " . . . .",
00513     " . . . .",
00514     " . . . .",
00515     " . . . .",
00516     " . . . .",
00517     " . . . .",
00518     " . . . .",
00519     " . . . .",
00520     " . . . .",
00521     " . . . .",
00522     " . . . .",
00523     " . . . .",
00524     " . . . .",
00525     " . . . .",
00526     " . . . .",
00527     " . . . .",
00528     " . . . .",
00529     " . . . .",
00530     " . . . .",
00531     " . . . .",
00532     " . . . .",
00533     " . . . .",
00534     " . . . .",
00535     " . . . .",
00536     " . . . .",
00537     " . . . .",
00538     " . . . .",
00539     " . . . .",
00540     " . . . .",
00541     " . . . .",
00542     " . . . .",
00543     " . . . .",
00544     " . . . .",
00545     " . . . .",
00546     " . . . .",
00547     " . . . .",
00548     " . . . .",
00549     " . . . .",
00550     " . . . .",
00551     " . . . .",
00552     " . . . .",
00553     " . . . .",
00554     " . . . .",
00555     " . . . .",
00556     " . . . .",
00557     " . . . .",
00558     " . . . .",
00559     " . . . .",
00560     " . . . .",
00561     " . . . .",
00562     " . . . .",
00563     " . . . .",
00564     " . . . .",
00565     " . . . .",
00566     " . . . .",
00567     " . . . .",
00568     " . . . .",
00569     " . . . .",
00570     " . . . .",
00571     " . . . .",
00572     " . . . .",
00573     " . . . .",
00574     " . . . .",
00575     " . . . .",
00576     " . . . .",
00577     " . . . .",
00578     " . . . .",
00579     " . . . .",
00580     " . . . .",
00581     " . . . .",
00582     " . . . .",
00583     " . . . .",
00584     " . . . .",
00585     " . . . .",
00586     " . . . .",
00587     " . . . .",
00588     " . . . .",
00589     " . . . .",
00590     " . . . .",
00591     " . . . .",
00592     " . . . .",
00593     " . . . .",
00594     " . . . .",
00595     " . . . .",
00596     " . . . .",
00597     " . . . .",
00598     " . . . .",
00599     " . . . .",
00600     " . . . .",
00601     " . . . .",
00602     " . . . .",
00603     " . . . .",
00604     " . . . .",
00605     " . . . .",
00606     " . . . .",
00607     " . . . .",
00608     " . . . .",
00609     " . . . .",
00610     " . . . .",
00611     " . . . .",
00612     " . . . .",
00613     " . . . .",
00614     " . . . .",
00615     " . . . .",
00616     " . . . .",
00617     " . . . .",
00618     " . . . .",
00619     " . . . .",
00620     " . . . .",
00621     " . . . .",
00622     " . . . .",
00623     " . . . .",
00624     " . . . .",
00625     " . . . .",
00626     " . . . .",
00627     " . . . .",
00628     " . . . .",
00629     " . . . .",
00630     " . . . .",
00631     " . . . .",
00632     " . . . .",
00633     " . . . .",
00634     " . . . .",
00635     " . . . .",
00636     " . . . .",
00637     " . . . .",
00638     " . . . .",
00639     " . . . .",
00640     " . . . .",
00641     " . . . .",
00642     " . . . .",
00643     " . . . .",
00644     " . . . .",
00645     " . . . .",
00646     " . . . .",
00647     " . . . .",
00648     " . . . .",
00649     " . . . .",
00650     " . . . .",
00651     " . . . .",
00652     " . . . .",
00653     " . . . .",
00654     " . . . .",
00655     " . . . .",
00656     " . . . .",
00657     " . . . .",
00658     " . . . .",
00659     " . . . .",
00660     " . . . .",
00661     " . . . .",
00662     " . . . .",
00663     " . . . .",
00664     " . . . .",
00665     " . . . .",
00666     " . . . .",
00667     " . . . .",
00668     " . . . .",
00669     " . . . .",
00670     " . . . .",
00671     " . . . .",
00672     " . . . .",
00673     " . . . .",
00674     " . . . .",
00675     " . . . .",
00676     " . . . .",
00677     " . . . .",
00678     " . . . .",
00679     " . . . .",
00680     " . . . .",
00681     " . . . .",
00682     " . . . .",
00683     " . . . .",
00684     " . . . .",
00685     " . . . .",
00686     " . . . .",
00687     " . . . .",
00688     " . . . .",
00689     " . . . .",
00690     " . . . .",
00691     " . . . .",
00692     " . . . .",
00693     " . . . .",
00694     " . . . .",
00695     " . . . .",
00696     " . . . .",
00697     " . . . .",
00698     " . . . .",
00699     " . . . .",
00700     " . . . .",
00701     " . . . .",
00702     " . . . .",
00703     " . . . .",
00704     " . . . .",
00705     " . . . .",
00706     " . . . .",
00707     " . . . .",
00708     " . . . .",
00709     " . . . .",
00710     " . . . .",
00711     " . . . .",
00712     " . . . .",
00713     " . . . .",
00714     " . . . .",
00715     " . . . .",
00716     " . . . .",
00717     " . . . .",
00718     " . . . .",
00719     " . . . .",
00720     " . . . .",
00721     " . . . .",
00722     " . . . .",
00723     " . . . .",
00724     " . . . .",
00725     " . . . .",
00726     " . . . .",
00727     " . . . .",
00728     " . . . .",
00729     " . . . .",
00730     " . . . .",
00731     " . . . .",
00732     " . . . .",
00733     " . . . .",
00734     " . . . .",
00735     " . . . .",
00736     " . . . .",
00737     " . . . .",
00738     " . . . .",
00739     " . . . .",
00740     " . . . .",
00741     " . . . .",
00742     " . . . .",
00743     " . . . .",
00744     " . . . .",
00745     " . . . .",
00746     " . . . .",
00747     " . . . .",
00748     " . . . .",
00749     " . . . .",
00750     " . . . .",
00751     " . . . .",
00752     " . . . .",
00753     " . . . .",
00754     " . . . .",
00755     " . . . .",
00756     " . . . .",
00757     " . . . .",
00758     " . . . .",
00759     " . . . .",
00760     " . . . .",
00761     " . . . .",
00762     " . . . .",
00763     " . . . .",
00764     " . . . .",
00765     " . . . .",
00766     " . . . .",
00767     " . . . .",
00768     " . . . .",
00769     " . . . .",
00770     " . . . .",
00771     " . . . .",
00772     " . . . .",
00773     " . . . .",
00774     " . . . .",
00775     " . . . .",
00776     " . . . .",
00777     " . . . .",
00778     " . . . .",
00779     " . . . .",
00780     " . . . .",
00781     " . . . .",
00782     " . . . .",
00783     " . . . .",
00784     " . . . .",
00785     " . . . .",
00786     " . . . .",
00787     " . . . .",
00788     " . . . .",
00789     " . . . .",
00790     " . . . .",
00791     " . . . .",
00792     " . . . .",
00793     " . . . .",
00794     " . . . .",
00795     " . . . .",
00796     " . . . .",
00797     " . . . .",
00798     " . . . .",
00799     " . . . .",
00800     " . . . .",
00801     " . . . .",
00802     " . . . .",
00803     " . . . .",
00804     " . . . .",
00805     " . . . .",
00806     " . . . .",
00807     " . . . .",
00808     " . . . .",
00809     " . . . .",
00810     " . . . .",
00811     " . . . .",
00812     " . . . .",
00813     " . . . .",
00814     " . . . .",
00815     " . . . .",
00816     " . . . .",
00817     " . . . .",
00818     " . . . .",
00819     " . . . .",
00820     " . . . .",
00821     " . . . .",
00822     " . . . .",
00823     " . . . .",
00824     " . . . .",
00825     " . . . .",
00826     " . . . .",
00827     " . . . .",
00828     " . . . .",
00829     " . . . .",
00830     " . . . .",
00831     " . . . .",
00832     " . . . .",
00833     " . . . .",
00834     " . . . .",
00835     " . . . .",
00836     " . . . .",
00837     " . . . .",
00838     " . . . .",
00839     " . . . .",
00840     " . . . .",
00841     " . . . .",
00842     " . . . .",
00843     " . . . .",
00844     " . . . .",
00845     " . . . .",
00846     " . . . .",
00847     " . . . .",
00848     " . . . .",
00849     " . . . .",
00850     " . . . .",
00851     " . . . .",
00852     " . . . .",
00853     " . . . .",
00854     " . . . .",
00855     " . . . .",
00856     " . . . .",
00857     " . . . .",
00858     " . . . .",
00859     " . . . .",
00860     " . . . .",
00861     " . . . .",
00862     " . . . .",
00863     " . . . .",
00864     " . . . .",
00865     " . . . .",
00866     " . . . .",
00867     " . . . .",
00868     " . . . .",
00869     " . . . .",
00870     " . . . .",
00871     " . . . .",
00872     " . . . .",
00873     " . . . .",
00874     " . . . .",
00875     " . . . .",
00876     " . . . .",
00877     " . . . .",
00878     " . . . .",
00879     " . . . .",
00880     " . . . .",
00881     " . . . .",
00882     " . . . .",
00883     " . . . .",
00884     " . . . .",
00885     " . . . .",
00886     " . . . .",
00887     " . . . .",
00888     " . . . .",
00889     " . . . .",
00890     " . . . .",
00891     " . . . .",
00892     " . . . .",
00893     " . . . .",
00894     " . . . .",
00895     " . . . .",
00896     " . . . .",
00897     " . . . .",
00898     " . . . .",
00899     " . . . .",
00900     " . . . .",
00901     " . . . .",
00902     " . . . .",
00903     " . . . .",
00904     " . . . .",
00905     " . . . .",
00906     " . . . .",
00907     " . . . .",
00908     " . . . .",
00909     " . . . .",
00910     " . . . .",
00911     " . . . .",
00912     " . . . .",
00913     " . . . .",
00914     " . . . .",
00915     " . . . .",
00916     " . . . .",
00917     " . . . .",
00918     " . . . .",
00919     " . . . .",
00920     " . . . .",
00921     " . . . .",
00922     " . . . .",
00923     " . . . .",
00924     " . . . .",
00925     " . . . .",
00926     " . . . .",
00927     " . . . .",
00928     " . . . .",
00929     " . . . .",
00930     " . . . .",
00931     " . . . .",
00932     " . . . .",
00933     " . . . .",
00934     " . . . .",
00935     " . . . .",
00936     " . . . .",
00937     " . . . .",
00938     " . . . .",
00939     " . . . .",
00940     " . . . .",
00941     " . .
```

```

00098 "    +++      .      .      +++  ",
00099 "    .      .      .      .      ",
00100 "    .      +++      .      .      ",
00101 "    .      +++++      .      .      ",
00102 "    .      +++++      .      .      ",
00103 "    .      +++++      .      .      ",
00104 "    .      +++      .      .      ",
00105 "    .      .      .      .      ",
00106 "    .      .      .      .      ",
00107 "    .      .      .      .      ",
00108 "    .      .      .      .      ",
00109 "    .      .      .      .      ",
00110 "    .      .      .      .      ",
00111 "    .      .      .      .      ",
00112 "    .      .      .      .      ",
00113 "    .      .      .      .      ",
00114 "    .      .      .      .      ",
00115 };
00116
00117 /*
00118 const char * logo[] = {
00119 "32 32 3 1",
00120 "  c #FFFFFFFFFFFF",
00121 ".  c #00000000FFFF",
00122 "X  c #FFFF00000000",
00123 "
00124 "
00125 "
00126 "    .      .      .      .      ",
00127 "    .      .      .      .      ",
00128 "    .      .      .      .      ",
00129 "    .      .      .      .      ",
00130 "    .      .      XXX      .      ",
00131 "    .      .      XXXXX      .      ",
00132 "    .      .      XXXXX      .      ",
00133 "    .      .      XXXXX      .      ",
00134 "    XXX      .      XXX      XXX      ",
00135 "    XXXXX      .      .      XXXXX      ",
00136 "    XXXXX      .      .      XXXXX      ",
00137 "    XXXXX      .      .      XXXXX      ",
00138 "    XXX      .      .      XXX      ",
00139 "    .      .      .      .      ",
00140 "    .      XXX      .      .      ",
00141 "    .      XXXXX      .      .      ",
00142 "    .      XXXXX      .      .      ",
00143 "    .      XXXXX      .      .      ",
00144 "    .      XXX      .      .      ",
00145 "    .      .      .      .      ",
00146 "    .      .      .      .      ",
00147 "    .      .      .      .      ",
00148 "    .      .      .      .      ",
00149 "    .      .      .      .      ",
00150 "    .      .      .      .      ",
00151 "    .      .      .      .      ",
00152 "
00153 "
00154 "    };
00155 */
00156
00157 Options options[1];
00159 Running running[1];
00161 Window window[1];
00163
00170 void
00171 input_save_direction_xml (xmlNode * node)
00172 {
00173     #if DEBUG_INTERFACE
00174     fprintf (stderr, "input_save_direction_xml: start\n");
00175     #endif
00176     if (input->nsteps)
00177     {
00178         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00179 input->nsteps);
00179         if (input->relaxation != DEFAULT_RELAXATION)
00180             xml_node_set_float (node, (const xmlChar *)
00181 LABEL_RELAXATION,
00182 input->relaxation);
00181         switch (input->direction)
00182         {
00183             {
00184                 case DIRECTION_METHOD_COORDINATES:
00185                     xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00186 (const xmlChar *) LABEL_COORDINATES);
00187                     break;
00188                 default:
00189                     xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00190 (const xmlChar *) LABEL_RANDOM);
00191                     xml_node_set_uint (node, (const xmlChar *)

```

```

    LABEL_NESTIMATES,
00192         input->nestimates);
00193     }
00194 }
00195 #if DEBUG_INTERFACE
00196 fprintf (stderr, "input_save_direction_xml: end\n");
00197 #endif
00198 }
00199
00200 void
00201 input_save_direction_json (JsonNode * node)
00202 {
00203     JsonObject *object;
00204     #if DEBUG_INTERFACE
00205     fprintf (stderr, "input_save_direction_json: start\n");
00206     #endif
00207     object = json_node_get_object (node);
00208     if (input->nsteps)
00209     {
00210         json_object_set_uint (object, LABEL_NSTEPS,
00211 input->nsteps);
00212         if (input->relaxation != DEFAULT_RELAXATION)
00213             json_object_set_float (object, LABEL_RELAXATION,
00214 input->relaxation);
00215         switch (input->direction)
00216         {
00217             case DIRECTION_METHOD_COORDINATES:
00218                 json_object_set_string_member (object, LABEL_DIRECTION,
00219 LABEL_COORDINATES);
00220                 break;
00221             default:
00222                 json_object_set_string_member (object, LABEL_DIRECTION,
00223 LABEL_RANDOM);
00224             json_object_set_uint (object, LABEL_NESTIMATES,
00225 input->nestimates);
00226         }
00227     }
00228     #if DEBUG_INTERFACE
00229     fprintf (stderr, "input_save_direction_json: end\n");
00230     #endif
00231 }
00232
00233 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     #if DEBUG_INTERFACE
00241     fprintf (stderr, "input_save_xml: start\n");
00242     #endif
00243     // Setting root XML node
00244     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00245     xmlDocSetRootElement (doc, node);
00246     // Adding properties to the root XML node
00247     if (xmlStrcmp
00248         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00249         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00250             (xmlChar *) input->result);
00251     if (xmlStrcmp
00252         ((const xmlChar *) input->variables, (const xmlChar *)
00253 variables_name))
00254         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00255             (xmlChar *) input->variables);
00256     file = g_file_new_for_path (input->directory);
00257     file2 = g_file_new_for_path (input->simulator);
00258     buffer = g_file_get_relative_path (file, file2);
00259     g_object_unref (file2);
00260     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00261     g_free (buffer);
00262     if (input->evaluator)
00263     {
00264         file2 = g_file_new_for_path (input->evaluator);
00265         buffer = g_file_get_relative_path (file, file2);
00266         g_object_unref (file2);
00267         if (xmlStrlen ((xmlChar *) buffer))
00268             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00269                 (xmlChar *) buffer);
00270         g_free (buffer);
00271     }
00272     if (input->seed != DEFAULT_RANDOM_SEED)
00273         xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
00274 input->seed);

```

```

00285
00286 // Setting the algorithm
00287 buffer = (char *) g_slice_alloc (64);
00288 switch (input->algorithm)
00289 {
00290     case ALGORITHM_MONTE_CARLO:
00291         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00292                     (const xmlChar *) LABEL_MONTE_CARLO);
00293         snprintf (buffer, 64, "%u", input->nsimulations);
00294         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00295                     (xmlChar *) buffer);
00296         snprintf (buffer, 64, "%u", input->niterations);
00297         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00298                     (xmlChar *) buffer);
00299         snprintf (buffer, 64, "%.3lg", input->tolerance);
00300         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00301                     (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%u", input->nbest);
00303         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304         input_save_direction_xml (node);
00305         break;
00306     case ALGORITHM_SWEEP:
00307         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                     (const xmlChar *) LABEL_SWEEP);
00309         snprintf (buffer, 64, "%u", input->niterations);
00310         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                     (xmlChar *) buffer);
00312         snprintf (buffer, 64, "%.3lg", input->tolerance);
00313         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00314                     (xmlChar *) buffer);
00315         snprintf (buffer, 64, "%u", input->nbest);
00316         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00317         input_save_direction_xml (node);
00318         break;
00319     default:
00320         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00321                     (const xmlChar *) LABEL_GENETIC);
00322         snprintf (buffer, 64, "%u", input->nsimulations);
00323         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00324                     (xmlChar *) buffer);
00325         snprintf (buffer, 64, "%u", input->niterations);
00326         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00327                     (xmlChar *) buffer);
00328         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00329         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00332                     (xmlChar *) buffer);
00333         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION,
00335                     (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
00344 // Setting the experimental data
00345 for (i = 0; i < input->nexperiments; ++i)
00346 {
00347     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00348     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00349                 (xmlChar *) input->experiment[i].name);
00350     if (input->experiment[i].weight != 1.)
00351         xml_node_set_float (child, (const xmlChar *)
00352 LABEL_WEIGHT,
00353                             input->experiment[i].weight);
00354     for (j = 0; j < input->experiment->ninputs; ++j)
00355         xmlSetProp (child, (const xmlChar *) template[j],
00356                     (xmlChar *) input->experiment[i].template[j]);
00357 }
00358
00359 // Setting the variables data
00360 for (i = 0; i < input->nvariables; ++i)
00361 {
00362     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00363     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00364                 (xmlChar *) input->variable[i].name);
00365     xml_node_set_float (child, (const xmlChar *)
00366 LABEL_MINIMUM,
00367                         input->variable[i].rangemin);
00368     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00369         xml_node_set_float (child, (const xmlChar *)
00370 LABEL_ABSOLUTE_MINIMUM,
00371                             input->variable[i].rangeminabs);

```

```

00368     xml_node_set_float (child, (const xmlChar *)
LABEL_MAXIMUM,
00369                         input->variable[i].rangemax);
00370     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00371         xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
DEFAULT_PRECISION)
00374         xml_node_set_uint (child, (const xmlChar *)
LABEL_PRECISION,
00375                             input->variable[i].precision);
00376     if (input->algorithm == ALGORITHM_SWEEP)
00377         xml_node_set_uint (child, (const xmlChar *)
LABEL_NSWEEPS,
00378                             input->variable[i].nsweeps);
00379     else if (input->algorithm == ALGORITHM_GENETIC)
00380         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00381                             input->variable[i].nbits);
00382     if (input->nsteps)
00383         xml_node_set_float (child, (const xmlChar *)
LABEL_STEP,
00384                             input->variable[i].step);
00385 }
00386
00387 // Saving the error norm
00388 switch (input->norm)
00389 {
00390     case ERROR_NORM_MAXIMUM:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                     (const xmlChar *) LABEL_MAXIMUM);
00393         break;
00394     case ERROR_NORM_P:
00395         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                     (const xmlChar *) LABEL_P);
00397         xml_node_set_float (node, (const xmlChar *) LABEL_P,
input->p);
00398         break;
00399     case ERROR_NORM_TAXICAB:
00400         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                     (const xmlChar *) LABEL_TAXICAB);
00402 }
00403
00404 #if DEBUG_INTERFACE
00405     fprintf (stderr, "input_save: end\n");
00406 #endif
00407 }
00408
00415 void
00416 input_save_json (JsonGenerator * generator)
00417 {
00418     unsigned int i, j;
00419     char *buffer;
00420     JsonNode *node, *child;
00421     JsonObject *object, *object2;
00422     JsonArray *array;
00423     GFile *file, *file2;
00424
00425 #if DEBUG_INTERFACE
00426     fprintf (stderr, "input_save_json: start\n");
00427 #endif
00428
00429     // Setting root JSON node
00430     node = json_node_new (JSON_NODE_OBJECT);
00431     object = json_node_get_object (node);
00432     json_generator_set_root (generator, node);
00433
00434     // Adding properties to the root JSON node
00435     if (strcmp (input->result, result_name))
00436         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00437     if (strcmp (input->variables, variables_name))
00438         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00439
00440     file = g_file_new_for_path (input->directory);
00441     file2 = g_file_new_for_path (input->simulator);
00442     buffer = g_file_get_relative_path (file, file2);
00443     g_object_unref (file2);
00444     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00445     g_free (buffer);
00446     if (input->evaluator)
00447     {
00448         file2 = g_file_new_for_path (input->evaluator);
00449         buffer = g_file_get_relative_path (file, file2);
00450         g_object_unref (file2);
00451         if (strlen (buffer))
00452             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);

```

```

00453     g_free (buffer);
00454 }
00455 if (input->seed != DEFAULT_RANDOM_SEED)
00456     json_object_set_uint (object, LABEL_SEED,
input->seed);
00457
00458 // Setting the algorithm
00459 buffer = (char *) g_slice_alloc (64);
00460 switch (input->algorithm)
00461 {
00462     case ALGORITHM_MONTE_CARLO:
00463         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00464         snprintf (buffer, 64, "%u", input->nsimulations);
00465         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00466         snprintf (buffer, 64, "%u", input->niterations);
00467         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00468         snprintf (buffer, 64, "%.3lg", input->tolerance);
00469         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00470         snprintf (buffer, 64, "%u", input->nbest);
00471         json_object_set_string_member (object, LABEL_NBEST, buffer);
00472         input_save_direction_json (node);
00473         break;
00474     case ALGORITHM_SWEEP:
00475         json_object_set_string_member (object, LABEL_ALGORITHM,
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_direction_json (node);
00483         break;
00484     default:
00485         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00486         snprintf (buffer, 64, "%u", input->nsimulations);
00487         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00488         snprintf (buffer, 64, "%u", input->niterations);
00489         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00490         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00491         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00492         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00493         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00494         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00495         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00496         break;
00497 }
00498 g_slice_free1 (64, buffer);
00499 if (input->threshold != 0.)
00500     json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00501
00502 // Setting the experimental data
00503 array = json_array_new ();
00504 for (i = 0; i < input->nexperiments; ++i)
00505 {
00506     child = json_node_new (JSON_NODE_OBJECT);
00507     object = json_node_get_object (child);
00508     json_object_set_string_member (object2, LABEL_NAME,
input->experiment[i].name);
00509     if (input->experiment[i].weight != 1.)
00510         json_object_set_float (object2, LABEL_WEIGHT,
input->experiment[i].weight);
00511     for (j = 0; j < input->experiment->ninputs; ++j)
00512         json_object_set_string_member (object2, template[j],
input->experiment[i].
template[j]);
00513     json_array_add_element (array, child);
00514 }
00515 json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00516
00517 // Setting the variables data
00518 array = json_array_new ();
00519 for (i = 0; i < input->nvariables; ++i)
00520 {
00521     child = json_node_new (JSON_NODE_OBJECT);
00522     object = json_node_get_object (child);
00523     json_object_set_string_member (object2, LABEL_NAME,
input->variable[i].name);
00524     json_object_set_float (object2, LABEL_MINIMUM,
input->variable[i].rangemin);
00525     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00526         json_object_set_float (object2,
LABEL_ABSOLUTE_MINIMUM,
input->variable[i].rangeminabs);
00527 }
00528

```

```

00534     json_object_set_float (object2, LABEL_MAXIMUM,
00535                           input->variable[i].rangemax);
00536     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537         json_object_set_float (object2,
00538                                LABEL_ABSOLUTE_MAXIMUM,
00539                                input->variable[i].rangemaxabs);
00540     if (input->variable[i].precision !=
00541         DEFAULT_PRECISION)
00542         json_object_set_uint (object2, LABEL_PRECISION,
00543                               input->variable[i].precision);
00544     if (input->algorithm == ALGORITHM_SWEEP)
00545         json_object_set_uint (object2, LABEL_NSWEEPS,
00546                               input->variable[i].nsweeps);
00547     else if (input->algorithm == ALGORITHM_GENETIC)
00548         json_object_set_uint (object2, LABEL_NBITS,
00549                               input->variable[i].nbits);
00550     if (input->nsteps)
00551         json_object_set_float (object, LABEL_STEP,
00552                                input->variable[i].step);
00553     json_array_add_element (array, child);
00554 }
00555 json_object_set_array_member (object, LABEL_VARIABLES, array);
00556 // Saving the error norm
00557 switch (input->norm)
00558 {
00559     case ERROR_NORM_MAXIMUM:
00560         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00561         break;
00562     case ERROR_NORM_P:
00563         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00564         json_object_set_float (object, LABEL_P, input->
00565                                p);
00566         break;
00567     case ERROR_NORM_TAXICAB:
00568         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00569 }
00570 #if DEBUG_INTERFACE
00571 fprintf (stderr, "input_save_json: end\n");
00572 #endif
00573 }
00574 void
00575 input_save (char *filename)
00576 {
00577     xmlDoc *doc;
00578     JsonGenerator *generator;
00579 #if DEBUG_INTERFACE
00580 fprintf (stderr, "input_save: start\n");
00581 #endif
00582 // Getting the input file directory
00583 input->name = g_path_get_basename (filename);
00584 input->directory = g_path_get_dirname (filename);
00585 if (input->type == INPUT_TYPE_XML)
00586 {
00587     // Opening the input file
00588     doc = xmlNewDoc ((const xmlChar *) "1.0");
00589     input_save_xml (doc);
00590     // Saving the XML file
00591     xmlSaveFormatFile (filename, doc, 1);
00592     // Freeing memory
00593     xmlFreeDoc (doc);
00594 }
00595 else
00596 {
00597     // Opening the input file
00598     generator = json_generator_new ();
00599     json_generator_set_pretty (generator, TRUE);
00600     input_save_json (generator);
00601     // Saving the JSON file
00602     json_generator_to_file (generator, filename, NULL);
00603     // Freeing memory
00604     g_object_unref (generator);
00605 }
00606 #if DEBUG_INTERFACE
00607 fprintf (stderr, "input_save: end\n");
00608 #endif
00609 }

```

```

00622
00627 void
00628 options_new ()
00629 {
00630     #if DEBUG_INTERFACE
00631         fprintf (stderr, "options_new: start\n");
00632     #endif
00633     options->label_seed = (GtkLabel *)
00634         gtk_label_new (_("Pseudo-random numbers generator seed"));
00635     options->spin_seed = (GtkSpinButton *)
00636         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00637     gtk_widget_set_tooltip_text
00638         (GTK_WIDGET (options->spin_seed),
00639          _("Seed to init the pseudo-random numbers generator"));
00640     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00641     options->label_threads = (GtkLabel *)
00642         gtk_label_new (_("Threads number for the stochastic algorithm"));
00643     options->spin_threads
00644         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00645     gtk_widget_set_tooltip_text
00646         (GTK_WIDGET (options->spin_threads),
00647          _("Number of threads to perform the calibration/optimization for "
00648            "the stochastic algorithm"));
00649     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00650     options->label_direction = (GtkLabel *)
00651         gtk_label_new (_("Threads number for the direction search method"));
00652     options->spin_direction =
00653         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00654     gtk_widget_set_tooltip_text (GTK_WIDGET (options->spin_direction),
00655         _("Number of threads to perform the calibration/optimization for "
00656           "the direction search method"));
00657     gtk_spin_button_set_value (options->spin_direction,
00658         (gdouble) nthreads_direction);
00659     options->grid = (GtkGrid *) gtk_grid_new ();
00660     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1,
00661         1);
00662     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1,
00663         1);
00664     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads), 0, 1,
00665         1, 1);
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads), 1, 1, 1,
00667         1);
00668     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction), 0, 2,
00669         1, 1);
00670     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction), 1, 2,
00671         1, 1);
00672     gtk_widget_show_all (GTK_WIDGET (options->grid));
00673     options->dialog = (GtkDialog *)
00674         gtk_dialog_new_with_buttons (_("Options"),
00675             window->window,
00676             GTK_DIALOG_MODAL,
00677             _("_OK"), GTK_RESPONSE_OK,
00678             _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00679     gtk_container_add
00680         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00681          GTK_WIDGET (options->grid));
00682     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00683     {
00684         input->seed
00685             = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00686         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00687         nthreads_direction
00688             = gtk_spin_button_get_value_as_int (options->spin_direction);
00689     }
00690     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00691     #if DEBUG_INTERFACE
00692         fprintf (stderr, "options_new: end\n");
00693     #endif
00694 }
00695
00701 void
00702 running_new ()
00703 {
00704     #if DEBUG_INTERFACE
00705         fprintf (stderr, "running_new: start\n");
00706     #endif
00707     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00708     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00709     running->grid = (GtkGrid *) gtk_grid_new ();
00710     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00711     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00712     running->dialog = (GtkDialog *)
00713         gtk_dialog_new_with_buttons (_("Calculating"),
00714             window->window, GTK_DIALOG_MODAL, NULL,

```



```

00715         NULL);
00716     gtk_container_add (GTK_CONTAINER
00717         (gtk_dialog_get_content_area (running->dialog)),
00718         GTK_WIDGET (running->grid));
00719     gtk_spinner_start (running->spinner);
00720     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721     #if DEBUG_INTERFACE
00722     fprintf (stderr, "running_new: end\n");
00723     #endif
00724 }
00725
00731 unsigned int
00732 window_get_algorithm ()
00733 {
00734     unsigned int i;
00735     #if DEBUG_INTERFACE
00736     fprintf (stderr, "window_get_algorithm: start\n");
00737     #endif
00738     i = gtk_array_get_active (window->button_algorithm,
00739         NALGORITHMS);
00739     #if DEBUG_INTERFACE
00740     fprintf (stderr, "window_get_algorithm: %u\n", i);
00741     fprintf (stderr, "window_get_algorithm: end\n");
00742     #endif
00743     return i;
00744 }
00745
00751 unsigned int
00752 window_get_direction ()
00753 {
00754     unsigned int i;
00755     #if DEBUG_INTERFACE
00756     fprintf (stderr, "window_get_direction: start\n");
00757     #endif
00758     i = gtk_array_get_active (window->button_direction,
00759         NDIRECTIONS);
00759     #if DEBUG_INTERFACE
00760     fprintf (stderr, "window_get_direction: %u\n", i);
00761     fprintf (stderr, "window_get_direction: end\n");
00762     #endif
00763     return i;
00764 }
00765
00771 unsigned int
00772 window_get_norm ()
00773 {
00774     unsigned int i;
00775     #if DEBUG_INTERFACE
00776     fprintf (stderr, "window_get_norm: start\n");
00777     #endif
00778     i = gtk_array_get_active (window->button_norm,
00779         NNORMS);
00779     #if DEBUG_INTERFACE
00780     fprintf (stderr, "window_get_norm: %u\n", i);
00781     fprintf (stderr, "window_get_norm: end\n");
00782     #endif
00783     return i;
00784 }
00785
00790 void
00791 window_save_direction ()
00792 {
00793     #if DEBUG_INTERFACE
00794     fprintf (stderr, "window_save_direction: start\n");
00795     #endif
00796     if (gtk_toggle_button_get_active
00797         (GTK_TOGGLE_BUTTON (window->check_direction)))
00798     {
00799         input->nsteps = gtk_spin_button_get_value_as_int (window->
00800 spin_steps);
00800         input->relaxation = gtk_spin_button_get_value (window->
00801 spin_relaxation);
00801         switch (window_get_direction ())
00802         {
00803             case DIRECTION_METHOD_COORDINATES:
00804                 input->direction = DIRECTION_METHOD_COORDINATES;
00805                 break;
00806             default:
00807                 input->direction = DIRECTION_METHOD_RANDOM;
00808                 input->nestimates
00809                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00810         }
00811     }
00812     else
00813         input->nsteps = 0;
00814     #if DEBUG_INTERFACE
00815     fprintf (stderr, "window_save_direction: end\n");

```

```

00816 #endif
00817 }
00818
00824 int
00825 window_save ()
00826 {
00827     GtkFileChooserDialog *dlg;
00828     GtkFileFilter *filter1, *filter2;
00829     char *buffer;
00830
00831     #if DEBUG_INTERFACE
00832     fprintf (stderr, "window_save: start\n");
00833     #endif
00834
00835     // Opening the saving dialog
00836     dlg = (GtkFileChooserDialog *)
00837         gtk_file_chooser_dialog_new (_("Save file"),
00838                                     window->window,
00839                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                     _("_Cancel"),
00841                                     GTK_RESPONSE_CANCEL,
00842                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00843     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844                                                    TRUE);
00845     buffer = g_build_filename (input->directory, input->name, NULL);
00846     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847     g_free (buffer);
00848
00849     // Adding XML filter
00850     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter1, "XML");
00852     gtk_file_filter_add_pattern (filter1, "*.xml");
00853     gtk_file_filter_add_pattern (filter1, "*.XML");
00854     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856     // Adding JSON filter
00857     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858     gtk_file_filter_set_name (filter2, "JSON");
00859     gtk_file_filter_add_pattern (filter2, "*.json");
00860     gtk_file_filter_add_pattern (filter2, "*.JSON");
00861     gtk_file_filter_add_pattern (filter2, "*.js");
00862     gtk_file_filter_add_pattern (filter2, "*.JS");
00863     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865     if (input->type == INPUT_TYPE_XML)
00866         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867     else
00868         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870     // If OK response then saving
00871     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872     {
00873         // Setting input file type
00874         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875         buffer = (char *) gtk_file_filter_get_name (filter1);
00876         if (!strcmp (buffer, "XML"))
00877             input->type = INPUT_TYPE_XML;
00878         else
00879             input->type = INPUT_TYPE_JSON;
00880
00881         // Adding properties to the root XML node
00882         input->simulator = gtk_file_chooser_get_filename
00883             (GTK_FILE_CHOOSER (window->button_simulator));
00884         if (gtk_toggle_button_get_active
00885             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886             input->evaluator = gtk_file_chooser_get_filename
00887                 (GTK_FILE_CHOOSER (window->button_evaluator));
00888         else
00889             input->evaluator = NULL;
00890         if (input->type == INPUT_TYPE_XML)
00891         {
00892             input->result
00893                 = (char *) xmlStrdup ((const xmlChar *)
00894                                         gtk_entry_get_text (window->entry_result));
00895             input->variables
00896                 = (char *) xmlStrdup ((const xmlChar *)
00897                                         gtk_entry_get_text
00898                                             (window->entry_variables));
00899         }
00900         else
00901         {
00902             input->result =
00903                 g_strdup (gtk_entry_get_text (window->entry_result));
00904             input->variables =
00905                 g_strdup (gtk_entry_get_text (window->entry_variables));
00906         }
00907

```

```

00908     // Setting the algorithm
00909     switch (window_get_algorithm ())
00910     {
00911         case ALGORITHM_MONTE_CARLO:
00912             input->algorithm = ALGORITHM_MONTE_CARLO;
00913             input->nsimulations
00914                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915             input->niterations
00916                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917             input->tolerance =
00918                 gtk_spin_button_get_value (window->spin_tolerance);
00919             input->nbest =
00920                 gtk_spin_button_get_value_as_int (window->spin_bests);
00921             window_save_direction ();
00922             break;
00923         case ALGORITHM_SWEEP:
00924             input->algorithm = ALGORITHM_SWEEP;
00925             input->niterations
00926                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927             input->tolerance =
00928                 gtk_spin_button_get_value (window->spin_tolerance);
00929             input->nbest =
00930                 gtk_spin_button_get_value_as_int (window->spin_bests);
00931             window_save_direction ();
00932             break;
00933         default:
00934             input->algorithm = ALGORITHM_GENETIC;
00935             input->nsimulations
00936                 = gtk_spin_button_get_value_as_int (window->spin_population);
00937             input->niterations
00938                 = gtk_spin_button_get_value_as_int (window->spin_generations);
00939             input->mutation_ratio
00940                 = gtk_spin_button_get_value (window->spin_mutation);
00941             input->reproduction_ratio
00942                 = gtk_spin_button_get_value (window->spin_reproduction);
00943             input->adaptation_ratio
00944                 = gtk_spin_button_get_value (window->spin_adaptation);
00945             break;
00946     }
00947     input->norm = window_get_norm ();
00948     input->p = gtk_spin_button_get_value (window->spin_p);
00949     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);

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 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

```

```

00998     gtk_spinner_stop (running->spinner);
00999     gtk_widget_destroy (GTK_WIDGET (running->dialog));
01000     #if DEBUG_INTERFACE
01001     fprintf (stderr, "window_run: displaying results\n");
01002     #endif
01003     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01004     msg2 = g_strdup (buffer);
01005     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01006     {
01007         snprintf (buffer, 64, "%s = %s\n",
01008                 input->variable[i].name,
01009                 format[input->variable[i].precision]);
01010         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01011         msg = g_strconcat (msg2, buffer2, NULL);
01012         g_free (msg2);
01013     }
01014     snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01015             optimize->calculation_time);
01016     msg = g_strconcat (msg2, buffer, NULL);
01017     g_free (msg2);
01018     show_message (_("Best result"), msg, INFO_TYPE);
01019     g_free (msg);
01020     #if DEBUG_INTERFACE
01021     fprintf (stderr, "window_run: freeing memory\n");
01022     #endif
01023     optimize_free ();
01024     #if DEBUG_INTERFACE
01025     fprintf (stderr, "window_run: end\n");
01026     #endif
01027 }
01028
01033 void
01034 window_help ()
01035 {
01036     char *buffer, *buffer2;
01037     #if DEBUG_INTERFACE
01038     fprintf (stderr, "window_help: start\n");
01039     #endif
01040     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01041                               _("user-manual.pdf"), NULL);
01042     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01043     g_free (buffer2);
01044     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
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
01058 void
01059 window_about ()
01060 {
01061     static const gchar *authors[] = {
01062         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01063         "Borja Latorre Garcés <borja.latorre@csic.es>",
01064         NULL
01065     };
01066     #if DEBUG_INTERFACE
01067     fprintf (stderr, "window_about: start\n");
01068     #endif
01069     gtk_show_about_dialog
01070     (window->window,
01071      "program_name", "MPCOTool",
01072      "comments",
01073      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01074        "A software to perform calibrations or optimizations of empirical"
01075        " parameters"),
01076      "authors", authors,
01077      "translator-credits",
01078      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01079      "(english, french and spanish)\n"
01080      "Uğur Çayoğlu (german)",
01081      "version", "3.4.0",
01082      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
01083      "logo", window->logo,
01084      "website", "https://github.com/jburguete/mpcotool",
01085      "license-type", GTK_LICENSE_BSD, NULL);
01086     #if DEBUG_INTERFACE
01087     fprintf (stderr, "window_about: end\n");
01088     #endif
01089 }
01090
01096 void
01097 window_update_direction ()

```

```

01098 {
01099     #if DEBUG_INTERFACE
01100         fprintf (stderr, "window_update_direction: start\n");
01101     #endif
01102     gtk_widget_show (GTK_WIDGET (window->check_direction));
01103     if (gtk_toggle_button_get_active
01104         (GTK_TOGGLE_BUTTON (window->check_direction)))
01105     {
01106         gtk_widget_show (GTK_WIDGET (window->grid_direction));
01107         gtk_widget_show (GTK_WIDGET (window->label_step));
01108         gtk_widget_show (GTK_WIDGET (window->spin_step));
01109     }
01110     switch (window_get_direction ())
01111     {
01112         case DIRECTION_METHOD_COORDINATES:
01113             gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01114             gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01115             break;
01116         default:
01117             gtk_widget_show (GTK_WIDGET (window->label_estimates));
01118             gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01119     }
01120     #if DEBUG_INTERFACE
01121         fprintf (stderr, "window_update_direction: end\n");
01122     #endif
01123 }
01124
01129 void
01130 window_update ()
01131 {
01132     unsigned int i;
01133     #if DEBUG_INTERFACE
01134         fprintf (stderr, "window_update: start\n");
01135     #endif
01136     gtk_widget_set_sensitive
01137         (GTK_WIDGET (window->button_evaluator),
01138          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01139                                         (window->check_evaluator)));
01140     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01142     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01144     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01146     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01148     gtk_widget_hide (GTK_WIDGET (window->label_population));
01149     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01150     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01151     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01152     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01153     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01154     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01155     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01156     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01157     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01158     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01159     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01160     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01161     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01162     gtk_widget_hide (GTK_WIDGET (window->check_direction));
01163     gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01164     gtk_widget_hide (GTK_WIDGET (window->label_step));
01165     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01166     gtk_widget_hide (GTK_WIDGET (window->label_p));
01167     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01168     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01169     switch (window_get_algorithm ())
01170     {
01171         case ALGORITHM_MONTE_CARLO:
01172             gtk_widget_show (GTK_WIDGET (window->label_simulations));
01173             gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01174             gtk_widget_show (GTK_WIDGET (window->label_iterations));
01175             gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01176             if (i > 1)
01177             {
01178                 gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01179                 gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01180                 gtk_widget_show (GTK_WIDGET (window->label_bests));
01181                 gtk_widget_show (GTK_WIDGET (window->spin_bests));
01182             }
01183             window_update_direction ();
01184             break;
01185         case ALGORITHM_SWEEP:
01186             gtk_widget_show (GTK_WIDGET (window->label_iterations));
01187             gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01188             if (i > 1)

```

```

01189     {
01190         gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01191         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01192         gtk_widget_show (GTK_WIDGET (window->label_bests));
01193         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01194     }
01195     gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01196     gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01197     gtk_widget_show (GTK_WIDGET (window->check_direction));
01198     window_update_direction ();
01199     break;
01200 default:
01201     gtk_widget_show (GTK_WIDGET (window->label_population));
01202     gtk_widget_show (GTK_WIDGET (window->spin_population));
01203     gtk_widget_show (GTK_WIDGET (window->label_generations));
01204     gtk_widget_show (GTK_WIDGET (window->spin_generations));
01205     gtk_widget_show (GTK_WIDGET (window->label_mutation));
01206     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01207     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01208     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01209     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01210     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01211     gtk_widget_show (GTK_WIDGET (window->label_bits));
01212     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01213 }
01214 gtk_widget_set_sensitive
01215 (GTK_WIDGET (window->button_remove_experiment),
input->nexperiments > 1);
01216 gtk_widget_set_sensitive
01217 (GTK_WIDGET (window->button_remove_variable), input->
nvariables > 1);
01218 for (i = 0; i < input->experiment->ninputs; ++i)
01219 {
01220     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01221     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01222     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01223     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01224     g_signal_handler_block
01225         (window->check_template[i], window->id_template[i]);
01226     g_signal_handler_block (window->button_template[i],
01227                             window->id_input[i]);
01228     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01229                                   (window->check_template[i]), 1);
01230     g_signal_handler_unblock (window->button_template[i],
01231                               window->id_input[i]);
01232     g_signal_handler_unblock (window->check_template[i],
01233                               window->id_template[i]);
01234 }
01235 if (i > 0)
01236 {
01237     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]),
01238                               1);
01239     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01240                               gtk_toggle_button_get_active
01241                                   (GTK_TOGGLE_BUTTON (window->check_template
01242                                                         [i - 1])));
01243 }
01244 if (i < MAX_NINPUTS)
01245 {
01246     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01247     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01248     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01249     gtk_widget_set_sensitive
01250         (GTK_WIDGET (window->button_template[i]),
01251          gtk_toggle_button_get_active
01252              (GTK_TOGGLE_BUTTON (window->check_template[i])));
01253     g_signal_handler_block
01254         (window->check_template[i], window->id_template[i]);
01255     g_signal_handler_block (window->button_template[i],
01256                             window->id_input[i]);
01257     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01258                                   (window->check_template[i]), 0);
01259     g_signal_handler_unblock (window->button_template[i],
01260                               window->id_input[i]);
01261     g_signal_handler_unblock (window->check_template[i],
01262                               window->id_template[i]);
01263 }
01264 while (++i < MAX_NINPUTS)
01265 {
01266     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01267     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01268 }
01269 gtk_widget_set_sensitive
01270 (GTK_WIDGET (window->spin_minabs),
01271  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01272 gtk_widget_set_sensitive
01273 (GTK_WIDGET (window->spin_maxabs),

```

```

01274     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs));
01275     if (window_get_norm () == ERROR_NORM_P)
01276     {
01277         gtk_widget_show (GTK_WIDGET (window->label_p));
01278         gtk_widget_show (GTK_WIDGET (window->spin_p));
01279     }
01280 #if DEBUG_INTERFACE
01281     fprintf (stderr, "window_update: end\n");
01282 #endif
01283 }
01284
01285 void
01290 window_set_algorithm ()
01291 {
01292     int i;
01293 #if DEBUG_INTERFACE
01294     fprintf (stderr, "window_set_algorithm: start\n");
01295 #endif
01296     i = window_get_algorithm ();
01297     switch (i)
01298     {
01299         case ALGORITHM_SWEEP:
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_sweeps,
01304                                       (gdouble) input->variable[i].
01305                                       nsweeps);
01306             break;
01307         case ALGORITHM_GENETIC:
01308             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01309             if (i < 0)
01310                 i = 0;
01311             gtk_spin_button_set_value (window->spin_bits,
01312                                       (gdouble) input->variable[i].nbits);
01313     }
01314     window_update ();
01315 #if DEBUG_INTERFACE
01316     fprintf (stderr, "window_set_algorithm: end\n");
01317 #endif
01318 }
01319
01320 void
01324 window_set_experiment ()
01325 {
01326     unsigned int i, j;
01327     char *buffer1, *buffer2;
01328 #if DEBUG_INTERFACE
01329     fprintf (stderr, "window_set_experiment: start\n");
01330 #endif
01331     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01332     gtk_spin_button_set_value (window->spin_weight,
01333                               input->experiment[i].weight);
01334     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01335     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01336     g_free (buffer1);
01337     g_signal_handler_block
01338         (window->button_experiment, window->id_experiment_name);
01339     gtk_file_chooser_set_filename
01340         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01341     g_signal_handler_unblock
01342         (window->button_experiment, window->id_experiment_name);
01343     g_free (buffer2);
01344     for (j = 0; j < input->experiment->ninputs; ++j)
01345     {
01346         g_signal_handler_block (window->button_template[j],
01347                               window->id_input[j]);
01348         buffer2 =
01349             g_build_filename (input->directory, input->experiment[i].
01350                             template[j],
01351                             NULL);
01352         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01353                                     (window->button_template[j]), buffer2);
01354         g_free (buffer2);
01355         g_signal_handler_unblock
01356             (window->button_template[j], window->id_input[j]);
01357     }
01358 #if DEBUG_INTERFACE
01359     fprintf (stderr, "window_set_experiment: end\n");
01360 #endif
01361 }
01362
01363 void
01367 window_remove_experiment ()
01368 {
01369     unsigned int i, j;
01370 #if DEBUG_INTERFACE

```

```

01371     fprintf (stderr, "window_remove_experiment: start\n");
01372 #endif
01373     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01374     g_signal_handler_block (window->combo_experiment, window->
01375         id_experiment);
01375     gtk_combo_box_text_remove (window->combo_experiment, i);
01376     g_signal_handler_unblock (window->combo_experiment, window->
01377         id_experiment);
01377     experiment_free (input->experiment + i, input->
01378         type);
01378     --input->nexperiments;
01379     for (j = i; j < input->nexperiments; ++j)
01380         memcpy (input->experiment + j, input->experiment + j + 1,
01381             sizeof (Experiment));
01382     j = input->nexperiments - 1;
01383     if (i > j)
01384         i = j;
01385     for (j = 0; j < input->experiment->ninputs; ++j)
01386         g_signal_handler_block (window->button_template[j], window->
01387             id_input[j]);
01387     g_signal_handler_block
01388         (window->button_experiment, window->id_experiment_name);
01389     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01390     g_signal_handler_unblock
01391         (window->button_experiment, window->id_experiment_name);
01392     for (j = 0; j < input->experiment->ninputs; ++j)
01393         g_signal_handler_unblock (window->button_template[j],
01394             window->id_input[j]);
01395     window_update ();
01396 #if DEBUG_INTERFACE
01397     fprintf (stderr, "window_remove_experiment: end\n");
01398 #endif
01399 }
01400
01405 void
01406 window_add_experiment ()
01407 {
01408     unsigned int i, j;
01409 #if DEBUG_INTERFACE
01410     fprintf (stderr, "window_add_experiment: start\n");
01411 #endif
01412     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01413     g_signal_handler_block (window->combo_experiment, window->
01414         id_experiment);
01414     gtk_combo_box_text_insert_text
01415         (window->combo_experiment, i, input->experiment[i].
01416             name);
01416     g_signal_handler_unblock (window->combo_experiment, window->
01417         id_experiment);
01417     input->experiment = (Experiment *) g_realloc
01418         (input->experiment, (input->nexperiments + 1) * sizeof (
01419             Experiment));
01419     for (j = input->nexperiments - 1; j > i; --j)
01420         memcpy (input->experiment + j + 1, input->experiment + j,
01421             sizeof (Experiment));
01422     input->experiment[j + 1].weight = input->experiment[j].
01423         weight;
01423     input->experiment[j + 1].ninputs = input->
01424         experiment[j].ninputs;
01424     if (input->type == INPUT_TYPE_XML)
01425     {
01426         input->experiment[j + 1].name
01427             = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
01428                 name);
01428         for (j = 0; j < input->experiment->ninputs; ++j)
01429             input->experiment[i + 1].template[j]
01430                 = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
01431                     template[j]);
01431     }
01432     else
01433     {
01434         input->experiment[j + 1].name = g_strdup (input->
01435             experiment[j].name);
01435         for (j = 0; j < input->experiment->ninputs; ++j)
01436             input->experiment[i + 1].template[j]
01437                 = g_strdup (input->experiment[i].template[j]);
01438     }
01439     ++input->nexperiments;
01440     for (j = 0; j < input->experiment->ninputs; ++j)
01441         g_signal_handler_block (window->button_template[j], window->
01442             id_input[j]);
01442     g_signal_handler_block
01443         (window->button_experiment, window->id_experiment_name);
01444     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01445     g_signal_handler_unblock
01446         (window->button_experiment, window->id_experiment_name);
01447     for (j = 0; j < input->experiment->ninputs; ++j)

```



```

01448     g_signal_handler_unblock (window->button_template[j],
01449                               window->id_input[j]);
01450     window_update ();
01451     #if DEBUG_INTERFACE
01452     fprintf (stderr, "window_add_experiment: end\n");
01453     #endif
01454 }
01455
01460 void
01461 window_name_experiment ()
01462 {
01463     unsigned int i;
01464     char *buffer;
01465     GFile *file1, *file2;
01466     #if DEBUG_INTERFACE
01467     fprintf (stderr, "window_name_experiment: start\n");
01468     #endif
01469     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01470     file1
01471     =
01472     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01473     file2 = g_file_new_for_path (input->directory);
01474     buffer = g_file_get_relative_path (file2, file1);
01475     g_signal_handler_block (window->combo_experiment, window->
01476                             id_experiment);
01477     gtk_combo_box_text_remove (window->combo_experiment, i);
01478     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01479     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01480     g_signal_handler_unblock (window->combo_experiment, window->
01481                             id_experiment);
01482     g_free (buffer);
01483     g_object_unref (file2);
01484     g_object_unref (file1);
01485     #if DEBUG_INTERFACE
01486     fprintf (stderr, "window_name_experiment: end\n");
01487     #endif
01488 }
01489
01492 void
01493 window_weight_experiment ()
01494 {
01495     unsigned int i;
01496     #if DEBUG_INTERFACE
01497     fprintf (stderr, "window_weight_experiment: start\n");
01498     #endif
01499     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01500     input->experiment[i].weight =
01501     gtk_spin_button_get_value (window->spin_weight);
01502     #if DEBUG_INTERFACE
01503     fprintf (stderr, "window_weight_experiment: end\n");
01504     #endif
01505 }
01506
01512 void
01513 window_inputs_experiment ()
01514 {
01515     unsigned int j;
01516     #if DEBUG_INTERFACE
01517     fprintf (stderr, "window_inputs_experiment: start\n");
01518     #endif
01519     j = input->experiment->ninputs - 1;
01520     if (j
01521         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01522                                           (window->check_template[j])))
01523         --input->experiment->ninputs;
01524     if (input->experiment->ninputs < MAX_NINPUTS
01525         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01526                                           (window->check_template[j])))
01527         ++input->experiment->ninputs;
01528     window_update ();
01529     #if DEBUG_INTERFACE
01530     fprintf (stderr, "window_inputs_experiment: end\n");
01531     #endif
01532 }
01533
01541 void
01542 window_template_experiment (void *data)
01543 {
01544     unsigned int i, j;
01545     char *buffer;
01546     GFile *file1, *file2;
01547     #if DEBUG_INTERFACE
01548     fprintf (stderr, "window_template_experiment: start\n");
01549     #endif
01550     i = (size_t) data;
01551     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552     file1

```

```

01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555     file2 = g_file_new_for_path (input->directory);
01556     buffer = g_file_get_relative_path (file2, file1);
01557     if (input->type == INPUT_TYPE_XML)
01558         input->experiment[j].template[i] =
01559             (char *) xmlStrdup ((xmlChar *) buffer);
01560     else
01561         input->experiment[j].template[i] = g_strdup (buffer);
01562     g_free (buffer);
01563     g_object_unref (file2);
01564     g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566     fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }
01569
01574 void
01575 window_set_variable ()
01576 {
01577     unsigned int i;
01578 #if DEBUG_INTERFACE
01579     fprintf (stderr, "window_set_variable: start\n");
01580 #endif
01581     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01582     g_signal_handler_block (window->entry_variable, window->
01583         id_variable_label);
01584     gtk_entry_set_text (window->entry_variable, input->variable[i].
01585         name);
01586     g_signal_handler_unblock (window->entry_variable,
01587         window->id_variable_label);
01588     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01589         rangemin);
01590     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01591         rangemax);
01592     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01593     {
01594         gtk_spin_button_set_value (window->spin_minabs,
01595             input->variable[i].rangeminabs);
01596         gtk_toggle_button_set_active
01597             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01598     }
01599     else
01600     {
01601         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01602         gtk_toggle_button_set_active
01603             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01604     }
01605     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01606     {
01607         gtk_spin_button_set_value (window->spin_maxabs,
01608             input->variable[i].rangemaxabs);
01609         gtk_toggle_button_set_active
01610             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01611     }
01612     else
01613     {
01614         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01615         gtk_toggle_button_set_active
01616             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01617     }
01618     gtk_spin_button_set_value (window->spin_precision,
01619         input->variable[i].precision);
01620     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01621         nsteps);
01622     if (input->nsteps)
01623         gtk_spin_button_set_value (window->spin_step, input->variable[i].
01624             step);
01625 #if DEBUG_INTERFACE
01626     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01627         input->variable[i].precision);
01628 #endif
01629     switch (window_get_algorithm ())
01630     {
01631     case ALGORITHM_SWEEP:
01632         gtk_spin_button_set_value (window->spin_sweeps,
01633             (gdouble) input->variable[i].
01634                 nsweeps);
01635 #if DEBUG_INTERFACE
01636     fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01637         input->variable[i].nsweeps);
01638 #endif
01639         break;
01640     case ALGORITHM_GENETIC:
01641         gtk_spin_button_set_value (window->spin_bits,
01642             (gdouble) input->variable[i].nbits);
01643 #if DEBUG_INTERFACE

```

```

01637         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01638                     input->variable[i].nbits);
01639     #endif
01640     break;
01641 }
01642 window_update ();
01643 #if DEBUG_INTERFACE
01644 fprintf (stderr, "window_set_variable: end\n");
01645 #endif
01646 }
01647
01652 void
01653 window_remove_variable ()
01654 {
01655     unsigned int i, j;
01656     #if DEBUG_INTERFACE
01657     fprintf (stderr, "window_remove_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->
01661                             id_variable);
01662     gtk_combo_box_text_remove (window->combo_variable, i);
01663     g_signal_handler_unblock (window->combo_variable, window->
01664                              id_variable);
01665     xmlFree (input->variable[i].name);
01666     --input->nvariables;
01667     for (j = i; j < input->nvariables; ++j)
01668         memcpy (input->variable + j, input->variable + j + 1, sizeof (
01669                 Variable));
01670     j = input->nvariables - 1;
01671     if (i > j)
01672         i = j;
01673     g_signal_handler_block (window->entry_variable, window->
01674                             id_variable_label);
01675     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01676     g_signal_handler_unblock (window->entry_variable,
01677                               window->id_variable_label);
01678     window_update ();
01679     #if DEBUG_INTERFACE
01680     fprintf (stderr, "window_remove_variable: end\n");
01681     #endif
01682 }
01683
01684 void
01685 window_add_variable ()
01686 {
01687     unsigned int i, j;
01688     #if DEBUG_INTERFACE
01689     fprintf (stderr, "window_add_variable: start\n");
01690     #endif
01691     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01692     g_signal_handler_block (window->combo_variable, window->
01693                             id_variable);
01694     gtk_combo_box_text_insert_text (window->combo_variable, i,
01695                                     input->variable[i].name);
01696     g_signal_handler_unblock (window->combo_variable, window->
01697                              id_variable);
01698     input->variable = (Variable *) g_realloc
01699         (input->variable, (input->nvariables + 1) * sizeof (
01700             Variable));
01701     for (j = input->nvariables - 1; j > i; --j)
01702         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01703             Variable));
01704     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01705             Variable));
01706     if (input->type == INPUT_TYPE_XML)
01707         input->variable[j + 1].name
01708             = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01709     else
01710         input->variable[j + 1].name = g_strdup (input->
01711             variable[j].name);
01712     ++input->nvariables;
01713     g_signal_handler_block (window->entry_variable, window->
01714                             id_variable_label);
01715     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01716     g_signal_handler_unblock (window->entry_variable,
01717                               window->id_variable_label);
01718     window_update ();
01719     #if DEBUG_INTERFACE
01720     fprintf (stderr, "window_add_variable: end\n");
01721     #endif
01722 }
01723
01724 void
01725 window_label_variable ()
01726 {
01727     unsigned int i;

```

```

01725     const char *buffer;
01726     #if DEBUG_INTERFACE
01727     fprintf (stderr, "window_label_variable: start\n");
01728     #endif
01729     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01730     buffer = gtk_entry_get_text (window->entry_variable);
01731     g_signal_handler_block (window->combo_variable, window->
01732     id_variable);
01732     gtk_combo_box_text_remove (window->combo_variable, i);
01733     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01734     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01735     g_signal_handler_unblock (window->combo_variable, window->
01736     id_variable);
01736     #if DEBUG_INTERFACE
01737     fprintf (stderr, "window_label_variable: end\n");
01738     #endif
01739 }
01740
01741 void
01742 window_precision_variable ()
01743 {
01744     unsigned int i;
01745     #if DEBUG_INTERFACE
01746     fprintf (stderr, "window_precision_variable: start\n");
01747     #endif
01748     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01749     input->variable[i].precision
01750     =
01751     (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01752     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
01753     precision);
01754     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
01755     precision);
01756     gtk_spin_button_set_digits (window->spin_minabs,
01757     input->variable[i].precision);
01758     gtk_spin_button_set_digits (window->spin_maxabs,
01759     input->variable[i].precision);
01760     #if DEBUG_INTERFACE
01761     fprintf (stderr, "window_precision_variable: end\n");
01762     #endif
01763 }
01764
01765 void
01766 window_rangemin_variable ()
01767 {
01768     unsigned int i;
01769     #if DEBUG_INTERFACE
01770     fprintf (stderr, "window_rangemin_variable: start\n");
01771     #endif
01772     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01773     input->variable[i].rangemin = gtk_spin_button_get_value (window->
01774     spin_min);
01775     #if DEBUG_INTERFACE
01776     fprintf (stderr, "window_rangemin_variable: end\n");
01777     #endif
01778 }
01779
01780 void
01781 window_rangemax_variable ()
01782 {
01783     unsigned int i;
01784     #if DEBUG_INTERFACE
01785     fprintf (stderr, "window_rangemax_variable: start\n");
01786     #endif
01787     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01788     input->variable[i].rangemax = gtk_spin_button_get_value (window->
01789     spin_max);
01790     #if DEBUG_INTERFACE
01791     fprintf (stderr, "window_rangemax_variable: end\n");
01792     #endif
01793 }
01794
01795 void
01796 window_rangeminabs_variable ()
01797 {
01798     unsigned int i;
01799     #if DEBUG_INTERFACE
01800     fprintf (stderr, "window_rangeminabs_variable: start\n");
01801     #endif
01802     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01803     input->variable[i].rangeminabs
01804     = gtk_spin_button_get_value (window->spin_minabs);
01805     #if DEBUG_INTERFACE
01806     fprintf (stderr, "window_rangeminabs_variable: end\n");
01807     #endif
01808 }
01809
01810
01811

```

```

01826 void
01827 window_rangemaxabs_variable ()
01828 {
01829     unsigned int i;
01830     #if DEBUG_INTERFACE
01831     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01832     #endif
01833     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01834     input->variable[i].rangemaxabs
01835     = gtk_spin_button_get_value (window->spin_maxabs);
01836     #if DEBUG_INTERFACE
01837     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01838     #endif
01839 }
01840
01841 void
01842 window_step_variable ()
01843 {
01844     unsigned int i;
01845     #if DEBUG_INTERFACE
01846     fprintf (stderr, "window_step_variable: start\n");
01847     #endif
01848     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01849     input->variable[i].step = gtk_spin_button_get_value (window->
01850     spin_step);
01851     #if DEBUG_INTERFACE
01852     fprintf (stderr, "window_step_variable: end\n");
01853     #endif
01854 }
01855
01856 void
01857 window_update_variable ()
01858 {
01859     int i;
01860     #if DEBUG_INTERFACE
01861     fprintf (stderr, "window_update_variable: start\n");
01862     #endif
01863     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01864     if (i < 0)
01865         i = 0;
01866     switch (window_get_algorithm ())
01867     {
01868     case ALGORITHM_SWEEP:
01869         input->variable[i].nsweeps
01870         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01871         #if DEBUG_INTERFACE
01872         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01873         input->variable[i].nsweeps);
01874         #endif
01875         break;
01876     case ALGORITHM_GENETIC:
01877         input->variable[i].nbits
01878         = gtk_spin_button_get_value_as_int (window->spin_bits);
01879         #if DEBUG_INTERFACE
01880         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01881         input->variable[i].nbits);
01882         #endif
01883     }
01884     #if DEBUG_INTERFACE
01885     fprintf (stderr, "window_update_variable: end\n");
01886     #endif
01887 }
01888
01889 int
01890 window_read (char *filename)
01891 {
01892     unsigned int i;
01893     char *buffer;
01894     #if DEBUG_INTERFACE
01895     fprintf (stderr, "window_read: start\n");
01896     #endif
01897     // Reading new input file
01898     input_free ();
01899     if (!input_open (filename))
01900     {
01901         #if DEBUG_INTERFACE
01902         fprintf (stderr, "window_read: end\n");
01903         #endif
01904         return 0;
01905     }
01906     // Setting GTK+ widgets data
01907     gtk_entry_set_text (window->entry_result, input->result);
01908     gtk_entry_set_text (window->entry_variables, input->
01909     variables);
01910     buffer = g_build_filename (input->directory, input->

```

```

    simulator, NULL);
01926   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01927   (window->button_simulator), buffer);
01928   g_free (buffer);
01929   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01930   (size_t) input->evaluator);
01931   if (input->evaluator)
01932   {
01933       buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01934       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01935       (window->button_evaluator), buffer);
01936       g_free (buffer);
01937   }
01938   gtk_toggle_button_set_active
01939   (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01940   switch (input->algorithm)
01941   {
01942       case ALGORITHM_MONTE_CARLO:
01943           gtk_spin_button_set_value (window->spin_simulations,
01944           (gdouble) input->nsimulations);
01945       case ALGORITHM_SWEEP:
01946           gtk_spin_button_set_value (window->spin_iterations,
01947           (gdouble) input->niterations);
01948           gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01949           gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01950           gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01951           (window->check_direction),
input->nsteps);
01952           if (input->nsteps)
01953           {
01954               gtk_toggle_button_set_active
01955               (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01956               gtk_spin_button_set_value (window->spin_steps,
01957               (gdouble) input->nsteps);
01958               gtk_spin_button_set_value (window->spin_relaxation,
01959               (gdouble) input->relaxation);
01960               switch (input->direction)
01961               {
01962                   case DIRECTION_METHOD_RANDOM:
01963                       gtk_spin_button_set_value (window->spin_estimates,
01964                       (gdouble) input->nestimates);
01965                   }
01966               }
01967               break;
01968           default:
01969               gtk_spin_button_set_value (window->spin_population,
01970               (gdouble) input->nsimulations);
01971               gtk_spin_button_set_value (window->spin_generations,
01972               (gdouble) input->niterations);
01973               gtk_spin_button_set_value (window->spin_mutation,
01974               input->mutation_ratio);
01975               gtk_spin_button_set_value (window->spin_reproduction,
01976               input->reproduction_ratio);
01977               gtk_spin_button_set_value (window->spin_adaptation,
01978               input->adaptation_ratio);
01979           }
01980       }
01981   gtk_toggle_button_set_active
01982   (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983   gtk_spin_button_set_value (window->spin_p, input->p);
01984   gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01985   g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01986   g_signal_handler_block (window->button_experiment,
01987   window->id_experiment_name);
01988   gtk_combo_box_text_remove_all (window->combo_experiment);
01989   for (i = 0; i < input->nexperiments; ++i)
01990       gtk_combo_box_text_append_text (window->combo_experiment,
01991       input->experiment[i].name);
01992   g_signal_handler_unblock
01993   (window->button_experiment, window->id_experiment_name);
01994   g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01995   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01996   g_signal_handler_block (window->combo_variable, window->
id_variable);
01997   g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01998   gtk_combo_box_text_remove_all (window->combo_variable);
01999   for (i = 0; i < input->nvariables; ++i)
02000       gtk_combo_box_text_append_text (window->combo_variable,
02001       input->variable[i].name);

```

```

02002 g_signal_handler_unblock (window->entry_variable,
02003                             window->id_variable_label);
02004 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
02005 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02006 window_set_variable ();
02007 window_update ();
02008
02009 #if DEBUG_INTERFACE
02010 fprintf (stderr, "window_read: end\n");
02011 #endif
02012 return 1;
02013 }
02014
02015 void
02020 window_open ()
02021 {
02022     GtkFileChooserDialog *dlg;
02023     GtkFileFilter *filter;
02024     char *buffer, *directory, *name;
02025
02026 #if DEBUG_INTERFACE
02027     fprintf (stderr, "window_open: start\n");
02028 #endif
02029
02030     // Saving a backup of the current input file
02031     directory = g_strdup (input->directory);
02032     name = g_strdup (input->name);
02033
02034     // Opening dialog
02035     dlg = (GtkFileChooserDialog *)
02036         gtk_file_chooser_dialog_new (_("Open input file"),
02037                                     window->window,
02038                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02039                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02040                                     _("_OK"), GTK_RESPONSE_OK, NULL);
02041
02042     // Adding XML filter
02043     filter = (GtkFileFilter *) gtk_file_filter_new ();
02044     gtk_file_filter_set_name (filter, "XML");
02045     gtk_file_filter_add_pattern (filter, "*.xml");
02046     gtk_file_filter_add_pattern (filter, "*.XML");
02047     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02048
02049     // Adding JSON filter
02050     filter = (GtkFileFilter *) gtk_file_filter_new ();
02051     gtk_file_filter_set_name (filter, "JSON");
02052     gtk_file_filter_add_pattern (filter, "*.json");
02053     gtk_file_filter_add_pattern (filter, "*.JSON");
02054     gtk_file_filter_add_pattern (filter, "*.js");
02055     gtk_file_filter_add_pattern (filter, "*.JS");
02056     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02057
02058     // If OK saving
02059     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02060     {
02061         // Trying to open the input file
02062         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02063         if (!window_read (buffer))
02064         {
02065             #if DEBUG_INTERFACE
02066                 fprintf (stderr, "window_open: error reading input file\n");
02067             #endif
02068             g_free (buffer);
02069
02070             // Reading backup file on error
02071             buffer = g_build_filename (directory, name, NULL);
02072             if (!input_open (buffer))
02073             {
02074                 // Closing on backup file reading error
02075
02076                 #if DEBUG_INTERFACE
02077                     fprintf (stderr, "window_read: error reading backup file\n");
02078                 #endif
02079                 g_free (buffer);
02080                 break;
02081             }
02082             g_free (buffer);
02083         }
02084     }
02085     else
02086     {
02087         g_free (buffer);
02088         break;
02089     }
02090 }
02091

```

```

02092 // Freeing and closing
02093 g_free (name);
02094 g_free (directory);
02095 gtk_widget_destroy (GTK_WIDGET (dlg));
02096 #if DEBUG_INTERFACE
02097 fprintf (stderr, "window_open: end\n");
02098 #endif
02099 }
02100
02107 void
02108 window_new (GtkApplication * application)
02109 {
02110     unsigned int i;
02111     char *buffer, *buffer2, buffer3[64];
02112     char *label_algorithm[NALGORITHMS] = {
02113         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114     };
02115     char *tip_algorithm[NALGORITHMS] = {
02116         _("Monte-Carlo brute force algorithm"),
02117         _("Sweep brute force algorithm"),
02118         _("Genetic algorithm")
02119     };
02120     char *label_direction[N DIRECTIONS] = {
02121         _("_Coordinates descent"), _("_Random")
02122     };
02123     char *tip_direction[N DIRECTIONS] = {
02124         _("Coordinates direction estimate method"),
02125         _("Random direction estimate method")
02126     };
02127     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128     char *tip_norm[NNORMS] = {
02129         _("Euclidean error norm (L2)"),
02130         _("Maximum error norm (L)"),
02131         _("P error norm (Lp)"),
02132         _("Taxicab error norm (L1)")
02133     };
02134
02135     #if DEBUG_INTERFACE
02136         fprintf (stderr, "window_new: start\n");
02137     #endif
02138
02139     // Creating the window
02140     window->window = main_window
02141         = (GtkWindow *) gtk_application_window_new (application);
02142
02143     // Finish when closing the window
02144     g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148     // Setting the window title
02149     gtk_window_set_title (window->window, "MPCOTool");
02150
02151     // Creating the open button
02152     window->button_open = (GtkToolButton *) gtk_tool_button_new
02153         (gtk_image_new_from_icon_name ("document-open",
02154                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157     // Creating the save button
02158     window->button_save = (GtkToolButton *) gtk_tool_button_new
02159         (gtk_image_new_from_icon_name ("document-save",
02160                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161     g_signal_connect (window->button_save, "clicked", (void *)
02162 window_save,
02163                     NULL);
02164
02165     // Creating the run button
02166     window->button_run = (GtkToolButton *) gtk_tool_button_new
02167         (gtk_image_new_from_icon_name ("system-run",
02168                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02169     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02170
02171     // Creating the options button
02172     window->button_options = (GtkToolButton *) gtk_tool_button_new
02173         (gtk_image_new_from_icon_name ("preferences-system",
02174                                     GTK_ICON_SIZE_LARGE_TOOLBAR),
02175         _("Options"));
02176     g_signal_connect (window->button_options, "clicked", options_new, NULL);
02177
02178     // Creating the help button
02179     window->button_help = (GtkToolButton *) gtk_tool_button_new
02180         (gtk_image_new_from_icon_name ("help-browser",
02181                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02182     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02183
02184     // Creating the about button

```



```

02184 window->button_about = (GtkToolButton *) gtk_tool_button_new
02185     (gtk_image_new_from_icon_name ("help-about",
02186     GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02187 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02188
02189 // Creating the exit button
02190 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02191     (gtk_image_new_from_icon_name ("application-exit",
02192     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02193 g_signal_connect_swapped (window->button_exit, "clicked",
02194     G_CALLBACK (g_application_quit),
02195     G_APPLICATION (application));
02196
02197 // Creating the buttons bar
02198 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02199 gtk_toolbar_insert
02200     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02201 gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02203 gtk_toolbar_insert
02204     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02205 gtk_toolbar_insert
02206     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02207 gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02209 gtk_toolbar_insert
02210     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02211 gtk_toolbar_insert
02212     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02213 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02214
02215 // Creating the simulator program label and entry
02216 window->label_simulator
02217     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02218 window->button_simulator = (GtkFileChooserButton *)
02219     gtk_file_chooser_button_new (_("Simulator program"),
02220     GTK_FILE_CHOOSER_ACTION_OPEN);
02221 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02222     _("Simulator program executable file"));
02223 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02224
02225 // Creating the evaluator program label and entry
02226 window->check_evaluator = (GtkCheckButton *)
02227     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02228 g_signal_connect (window->check_evaluator, "toggled",
02229     window_update, NULL);
02230 window->button_evaluator = (GtkFileChooserButton *)
02231     gtk_file_chooser_button_new (_("Evaluator program"),
02232     GTK_FILE_CHOOSER_ACTION_OPEN);
02233 gtk_widget_set_tooltip_text
02234     (GTK_WIDGET (window->button_evaluator),
02235     _("Optional evaluator program executable file"));
02236
02237 // Creating the results files labels and entries
02238 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02239 window->entry_result = (GtkEntry *) gtk_entry_new ();
02240 gtk_widget_set_tooltip_text
02241     (GTK_WIDGET (window->entry_result), _("Best results file"));
02242 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02243 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02244 gtk_widget_set_tooltip_text
02245     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02246
02247 // Creating the files grid and attaching widgets
02248 window->grid_files = (GtkGrid *) gtk_grid_new ();
02249 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02250     label_simulator),
02251     0, 0, 1, 1);
02252 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02253     button_simulator),
02254     1, 0, 1, 1);
02255 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02256     check_evaluator),
02257     0, 1, 1, 1);
02258 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02259     button_evaluator),
02260     1, 1, 1, 1);
02261 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02262     label_result),
02263     0, 2, 1, 1);
02264 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02265     entry_result),
02266     1, 2, 1, 1);
02267 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02268     label_variables),
02269     0, 3, 1, 1);
02270 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->

```

```

    entry_variables),
02263         1, 3, 1, 1);
02264
02265 // Creating the algorithm properties
02266 window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268 window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270 gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272      _("Number of simulations to perform for each iteration"));
02273 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274 window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276 window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278 gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280 g_signal_connect
02281     (window->spin_iterations, "value-changed", window_update, NULL);
02282 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284 window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286 gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288      _("Tolerance to set the variable interval on the next iteration"));
02289 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290 window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292 gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294      _("Number of best simulations used to set the variable interval "
02295        "on the next iteration"));
02296 window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298 window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300 gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302      _("Number of population for the genetic algorithm"));
02303 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304 window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306 window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308 gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310      _("Number of generations for the genetic algorithm"));
02311 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312 window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314 gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316      _("Ratio of mutation for the genetic algorithm"));
02317 window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319 window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321 gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323      _("Ratio of reproduction for the genetic algorithm"));
02324 window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326 window->spin_adaptation
02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328 gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332 window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335 gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338 window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                   GTK_WIDGET (window->spin_threshold));
02342 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344 //                          GTK_ALIGN_FILL);
02345
02346 // Creating the direction search method properties
02347 window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("_Direction search method"));

```

```

02349 g_signal_connect (window->check_direction, "clicked",
window_update, NULL);
02350 window->grid_direction = (GtkGrid *) gtk_grid_new ();
02351 window->button_direction[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02353 gtk_grid_attach (window->grid_direction,
02354     GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02355 g_signal_connect (window->button_direction[0], "clicked",
window_update,
02356     NULL);
02357 for (i = 0; ++i < NDIRECTIONS;)
02358 {
02359     window->button_direction[i] = (GtkRadioButton *)
02360     gtk_radio_button_new_with_mnemonic
02361     (gtk_radio_button_get_group (window->button_direction[0]),
02362     label_direction[i]);
02363     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02364     tip_direction[i]);
02365     gtk_grid_attach (window->grid_direction,
02366     GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02367     g_signal_connect (window->button_direction[i], "clicked",
02368     window_update, NULL);
02369 }
02370 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02371 window->spin_steps = (GtkSpinButton *)
02372     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02373 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02374 window->label_estimates
02375     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02376 window->spin_estimates = (GtkSpinButton *)
02377     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02378 window->label_relaxation
02379     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02380 window->spin_relaxation = (GtkSpinButton *)
02381     gtk_spin_button_new_with_range (0., 2., 0.001);
02382 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
label_steps),
02383     0, NDIRECTIONS, 1, 1);
02384 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
spin_steps),
02385     1, NDIRECTIONS, 1, 1);
02386 (window->grid_direction,
02387     GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02388     1, 1);
02389 (window->grid_direction,
02390     GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02391     1);
02392 (window->grid_direction,
02393     GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02394     1, 1);
02395 (window->grid_direction,
02396     GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02397     1, 1);
02398
02399 // Creating the array of algorithms
02400 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02401 window->button_algorithm[0] = (GtkRadioButton *)
02402     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02403 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02404     tip_algorithm[0]);
02405 gtk_grid_attach (window->grid_algorithm,
02406     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02407 g_signal_connect (window->button_algorithm[0], "clicked",
02408     window_set_algorithm, NULL);
02409 for (i = 0; ++i < NALGORITHMS;)
02410 {
02411     window->button_algorithm[i] = (GtkRadioButton *)
02412     gtk_radio_button_new_with_mnemonic
02413     (gtk_radio_button_get_group (window->button_algorithm[0]),
02414     label_algorithm[i]);
02415     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416     tip_algorithm[i]);
02417     gtk_grid_attach (window->grid_algorithm,
02418     GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419     g_signal_connect (window->button_algorithm[i], "clicked",
02420     window_set_algorithm, NULL);
02421 }
02422 gtk_grid_attach (window->grid_algorithm,
02423     GTK_WIDGET (window->label_simulations), 0,
02424     NALGORITHMS, 1, 1);
02425 gtk_grid_attach (window->grid_algorithm,
02426     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427     1);
02428 gtk_grid_attach (window->grid_algorithm,
02429     GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430     1, 1);
02431 gtk_grid_attach (window->grid_algorithm,

```

```

02432         GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433         1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435         GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436         1, 1);
02437     gtk_grid_attach (window->grid_algorithm,
02438         GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439         1);
02440     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_bests),
02441         0, NALGORITHMS + 3, 1, 1);
02442     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_bests), 1,
02443         NALGORITHMS + 3, 1, 1);
02444     gtk_grid_attach (window->grid_algorithm,
02445         GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02446         1, 1);
02447     gtk_grid_attach (window->grid_algorithm,
02448         GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02449         1, 1);
02450     gtk_grid_attach (window->grid_algorithm,
02451         GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02452         1, 1);
02453     gtk_grid_attach (window->grid_algorithm,
02454         GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02455         1, 1);
02456     gtk_grid_attach (window->grid_algorithm,
02457         GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02458         1);
02459     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
spin_mutation),
02460         1, NALGORITHMS + 6, 1, 1);
02461     gtk_grid_attach (window->grid_algorithm,
02462         GTK_WIDGET (window->label_reproduction), 0,
02463         NALGORITHMS + 7, 1, 1);
02464     gtk_grid_attach (window->grid_algorithm,
02465         GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02466         1, 1);
02467     gtk_grid_attach (window->grid_algorithm,
02468         GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02469         1, 1);
02470     gtk_grid_attach (window->grid_algorithm,
02471         GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02472         1, 1);
02473     gtk_grid_attach (window->grid_algorithm,
02474         GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02475         2, 1);
02476     gtk_grid_attach (window->grid_algorithm,
02477         GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478         2, 1);
02479     gtk_grid_attach (window->grid_algorithm,
02480         GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481         1, 1);
02482     gtk_grid_attach (window->grid_algorithm,
02483         GTK_WIDGET (window->scrolled_threshold), 1,
02484         NALGORITHMS + 11, 1, 1);
02485     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487         GTK_WIDGET (window->grid_algorithm));
02488
02489     // Creating the variable widgets
02490     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491     gtk_widget_set_tooltip_text
02492         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493     window->id_variable = g_signal_connect
02494         (window->combo_variable, "changed", window_set_variable, NULL);
02495     window->button_add_variable
02496         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497             GTK_ICON_SIZE_BUTTON);
02498     g_signal_connect
02499         (window->button_add_variable, "clicked",
02500         window_add_variable, NULL);
02501     gtk_widget_set_tooltip_text
02502         (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02503     window->button_remove_variable
02504         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02505             GTK_ICON_SIZE_BUTTON);
02506     g_signal_connect
02507         (window->button_remove_variable, "clicked",
02508         window_remove_variable, NULL);
02509     gtk_widget_set_tooltip_text
02510         (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02511     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02512     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02513     gtk_widget_set_tooltip_text
02514         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02515     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);

```

```

02514 window->id_variable_label = g_signal_connect
02515 (window->entry_variable, "changed", window_label_variable, NULL);
02516 window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02517 window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02518 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02519 gtk_widget_set_tooltip_text
02520 (GTK_WIDGET (window->spin_min),
02521 _("Minimum initial value of the variable"));
02522 window->scrolled_min
02523 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02524 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02525 GTK_WIDGET (window->spin_min));
02526 g_signal_connect (window->spin_min, "value-changed",
02527 window_rangemin_variable, NULL);
02528 window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02529 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02530 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02531 gtk_widget_set_tooltip_text
02532 (GTK_WIDGET (window->spin_max),
02533 _("Maximum initial value of the variable"));
02534 window->scrolled_max
02535 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02536 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02537 GTK_WIDGET (window->spin_max));
02538 g_signal_connect (window->spin_max, "value-changed",
02539 window_rangemax_variable, NULL);
02540 window->check_minabs = (GtkCheckButton *)
02541 gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02542 g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02543 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02544 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02545 gtk_widget_set_tooltip_text
02546 (GTK_WIDGET (window->spin_minabs),
02547 _("Minimum allowed value of the variable"));
02548 window->scrolled_minabs
02549 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02550 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02551 GTK_WIDGET (window->spin_minabs));
02552 g_signal_connect (window->spin_minabs, "value-changed",
02553 window_rangeminabs_variable, NULL);
02554 window->check_maxabs = (GtkCheckButton *)
02555 gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02556 g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02557 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02558 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02559 gtk_widget_set_tooltip_text
02560 (GTK_WIDGET (window->spin_maxabs),
02561 _("Maximum allowed value of the variable"));
02562 window->scrolled_maxabs
02563 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02564 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02565 GTK_WIDGET (window->spin_maxabs));
02566 g_signal_connect (window->spin_maxabs, "value-changed",
02567 window_rangemaxabs_variable, NULL);
02568 window->label_precision
02569 = (GtkLabel *) gtk_label_new (_("Precision digits"));
02570 window->spin_precision = (GtkSpinButton *)
02571 gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02572 gtk_widget_set_tooltip_text
02573 (GTK_WIDGET (window->spin_precision),
02574 _("Number of precision floating point digits\n"
02575 "0 is for integer numbers"));
02576 g_signal_connect (window->spin_precision, "value-changed",
02577 window_precision_variable, NULL);
02578 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579 window->spin_sweeps =
02580 (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582 _("Number of steps sweeping the variable"));
02583 g_signal_connect (window->spin_sweeps, "value-changed",
02584 window_update_variable, NULL);
02585 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586 window->spin_bits
02587 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588 gtk_widget_set_tooltip_text
02589 (GTK_WIDGET (window->spin_bits),
02590 _("Number of bits to encode the variable"));
02591 g_signal_connect
02592 (window->spin_bits, "value-changed", window_update_variable, NULL);
02593 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596 gtk_widget_set_tooltip_text
02597 (GTK_WIDGET (window->spin_step),
02598 _("Initial step size for the direction search method"));
02599 window->scrolled_step
02600 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);

```

```

02601 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602 GTK_WIDGET (window->spin_step));
02603 g_signal_connect
02604 (window->spin_step, "value-changed", window_step_variable, NULL);
02605 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606 gtk_grid_attach (window->grid_variable,
02607 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608 gtk_grid_attach (window->grid_variable,
02609 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610 gtk_grid_attach (window->grid_variable,
02611 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614 gtk_grid_attach (window->grid_variable,
02615 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616 gtk_grid_attach (window->grid_variable,
02617 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618 gtk_grid_attach (window->grid_variable,
02619 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620 gtk_grid_attach (window->grid_variable,
02621 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622 gtk_grid_attach (window->grid_variable,
02623 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624 gtk_grid_attach (window->grid_variable,
02625 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626 gtk_grid_attach (window->grid_variable,
02627 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628 gtk_grid_attach (window->grid_variable,
02629 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630 gtk_grid_attach (window->grid_variable,
02631 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632 gtk_grid_attach (window->grid_variable,
02633 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634 gtk_grid_attach (window->grid_variable,
02635 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636 gtk_grid_attach (window->grid_variable,
02637 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638 gtk_grid_attach (window->grid_variable,
02639 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640 gtk_grid_attach (window->grid_variable,
02641 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642 gtk_grid_attach (window->grid_variable,
02643 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644 gtk_grid_attach (window->grid_variable,
02645 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646 gtk_grid_attach (window->grid_variable,
02647 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650 GTK_WIDGET (window->grid_variable));
02651
02652 // Creating the experiment widgets
02653 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655 _("Experiment selector"));
02656 window->id_experiment = g_signal_connect
02657 (window->combo_experiment, "changed", window_set_experiment, NULL);
02658 ;
02659 window->button_add_experiment
02660 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02661 GTK_ICON_SIZE_BUTTON);
02662 g_signal_connect
02663 (window->button_add_experiment, "clicked",
02664 window_add_experiment, NULL);
02665 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02666 _("Add experiment"));
02667 window->button_remove_experiment
02668 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02669 GTK_ICON_SIZE_BUTTON);
02670 g_signal_connect (window->button_remove_experiment, "clicked",
02671 window_remove_experiment, NULL);
02672 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02673 _("Remove experiment"));
02674 window->label_experiment
02675 = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02676 window->button_experiment = (GtkFileChooserButton *)
02677 gtk_file_chooser_button_new (_("Experimental data file"),
02678 GTK_FILE_CHOOSER_ACTION_OPEN);
02679 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02680 _("Experimental data file"));
02681 window->id_experiment_name
02682 = g_signal_connect (window->button_experiment, "selection-changed",
02683 window_name_experiment, NULL);
02684 gtk_widget_set_hexpend (GTK_WIDGET (window->button_experiment), TRUE);
02685 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02686 window->spin_weight
02687 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);

```



```

02686 gtk_widget_set_tooltip_text
02687 (GTK_WIDGET (window->spin_weight),
02688 _("Weight factor to build the objective function"));
02689 g_signal_connect
02690 (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
02691 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692 gtk_grid_attach (window->grid_experiment,
02693 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694 gtk_grid_attach (window->grid_experiment,
02695 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696 gtk_grid_attach (window->grid_experiment,
02697 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02698 gtk_grid_attach (window->grid_experiment,
02699 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700 gtk_grid_attach (window->grid_experiment,
02701 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702 gtk_grid_attach (window->grid_experiment,
02703 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704 gtk_grid_attach (window->grid_experiment,
02705 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706 for (i = 0; i < MAX_NINPUTS; ++i)
02707 {
02708     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709     window->check_template[i] = (GtkCheckButton *)
02710     gtk_check_button_new_with_label (buffer3);
02711     window->id_template[i]
02712     = g_signal_connect (window->check_template[i], "toggled",
02713     window_inputs_experiment, NULL);
02714     gtk_grid_attach (window->grid_experiment,
02715     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716     1);
02717     window->button_template[i] =
02718     (GtkFileChooserButton *)
02719     gtk_file_chooser_button_new (_("Input template"),
02720     GTK_FILE_CHOOSER_ACTION_OPEN);
02721     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722     _("Experimental input template file"));
02723     window->id_input[i] =
02724     g_signal_connect_swapped (window->button_template[i],
02725     "selection-changed",
02726     (void (*)(void *)) window_template_experiment,
02727     (void *) (size_t) i);
02728     gtk_grid_attach (window->grid_experiment,
02729     GTK_WIDGET (window->button_template[i]),
02730     1, 3 + i, 3, 1);
02731 }
02732 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734     GTK_WIDGET (window->grid_experiment));
02735
02736 // Creating the error norm widgets
02737 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740     GTK_WIDGET (window->grid_norm));
02741 window->button_norm[0] = (GtkRadioButton *)
02742     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02744     tip_norm[0]);
02745     gtk_grid_attach (window->grid_norm,
02746     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747     g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02748     for (i = 0; ++i < NNORMS;)
02749     {
02750         window->button_norm[i] = (GtkRadioButton *)
02751         gtk_radio_button_new_with_mnemonic
02752         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02753         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02754         tip_norm[i]);
02755         gtk_grid_attach (window->grid_norm,
02756         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02757         g_signal_connect (window->button_norm[i], "clicked",
window_update,
02758             NULL);
02759     }
02760     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02761     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1,
02762     1);
02763     window->spin_p =
02764     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02765     G_MAXDOUBLE, 0.01);
02766     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02767     _("P parameter for the P error norm"));
02768     window->scrolled_p =
02769     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02770     gtk_container_add (GTK_CONTAINER (window->scrolled_p),

```

```

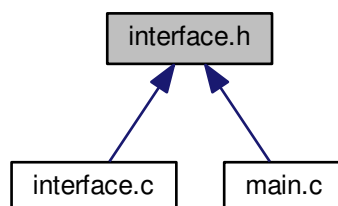
02771         GTK_WIDGET (window->spin_p));
02772 gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02773 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02774 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02775                 1, 2, 1, 2);
02776
02777 // Creating the grid and attaching the widgets to the grid
02778 window->grid = (GtkGrid *) gtk_grid_new ();
02779 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02780                 1);
02781 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02782 gtk_grid_attach (window->grid,
02783                 GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02784 gtk_grid_attach (window->grid,
02785                 GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02786 gtk_grid_attach (window->grid,
02787                 GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02788 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02789 gtk_container_add (GTK_CONTAINER (window->window),
02790                 GTK_WIDGET (window->grid));
02791
02792 // Setting the window logo
02793 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02794 gtk_window_set_icon (window->window, window->logo);
02795
02796 // Showing the window
02797 gtk_widget_show_all (GTK_WIDGET (window->window));
02798
02799 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02800 #if GTK_MINOR_VERSION >= 16
02801 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02802 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02803 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02804 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02805 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02806 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02807 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02808                             40);
02809 #endif
02810
02811 // Reading initial example
02812 input_new ();
02813 buffer2 = g_get_current_dir ();
02814 buffer =
02815     g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02816 g_free (buffer2);
02817 window_read (buffer);
02818 g_free (buffer);
02819
02820 #if DEBUG_INTERFACE
02821 fprintf (stderr, "window_new: start\n");
02822 #endif
02823 }

```

4.13 interface.h File Reference

Header file to define the graphical interface functions.

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



Data Structures

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

Macros

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

Functions

- unsigned int [gtk_array_get_active](#) (GtkRadioButton *array[], unsigned int n)
Function to get the active GtkRadioButton.
- void [input_save](#) (char *filename)
Function to save the input file.
- void [options_new](#) ()
Function to open the options dialog.
- void [running_new](#) ()
Function to open the running dialog.
- unsigned int [window_get_algorithm](#) ()
Function to get the stochastic algorithm number.
- unsigned int [window_get_direction](#) ()
Function to get the direction search method number.
- unsigned int [window_get_norm](#) ()
Function to get the norm method number.
- void [window_save_direction](#) ()
Function to save the direction search method data in the input file.
- int [window_save](#) ()
Function to save the input file.
- void [window_run](#) ()
Function to run a optimization.
- void [window_help](#) ()
Function to show a help dialog.
- void [window_update_direction](#) ()
Function to update direction search method widgets view in the main window.
- void [window_update](#) ()
Function to update the main window view.
- void [window_set_algorithm](#) ()
Function to avoid memory errors changing the algorithm.
- void [window_set_experiment](#) ()
Function to set the experiment data in the main window.
- void [window_remove_experiment](#) ()
Function to remove an experiment in the main window.
- void [window_add_experiment](#) ()

- Function to add an experiment in the main window.*

 - void `window_name_experiment ()`

Function to set the experiment name in the main window.
- void `window_weight_experiment ()`

Function to update the experiment weight in the main window.
- void `window_inputs_experiment ()`

Function to update the experiment input templates number in the main window.
- void `window_template_experiment (void *data)`

Function to update the experiment i-th input template in the main window.
- void `window_set_variable ()`

Function to set the variable data in the main window.
- void `window_remove_variable ()`

Function to remove a variable in the main window.
- void `window_add_variable ()`

Function to add a variable in the main window.
- void `window_label_variable ()`

Function to set the variable label in the main window.
- void `window_precision_variable ()`

Function to update the variable precision in the main window.
- void `window_rangemin_variable ()`

Function to update the variable rangemin in the main window.
- void `window_rangemax_variable ()`

Function to update the variable rangemax in the main window.
- void `window_rangeminabs_variable ()`

Function to update the variable rangeminabs in the main window.
- void `window_rangemaxabs_variable ()`

Function to update the variable rangemaxabs in the main window.
- void `window_update_variable ()`

Function to update the variable data in the main window.
- int `window_read (char *filename)`

Function to read the input data of a file.
- void `window_open ()`

Function to open the input data.
- void `window_new (GtkApplication *application)`

Function to open the main window.

Variables

- const char * `logo []`

Logo pixmap.
- `Options options [1]`

Options struct to define the options dialog.
- `Running running [1]`

Running struct to define the running dialog.
- `Window window [1]`

Window struct to define the main interface window.

4.13.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.13.2 Function Documentation

4.13.2.1 gtk_array_get_active()

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

Function to get the active GtkRadioButton.

Parameters

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

Returns

Active GtkRadioButton.

Definition at line [567](#) of file [utils.c](#).

```
00568 {  
00569     unsigned int i;  
00570     for (i = 0; i < n; ++i)  
00571         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))  
00572             break;  
00573     return i;  
00574 }
```

4.13.2.2 input_save()

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

Function to save the input file.

Parameters

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

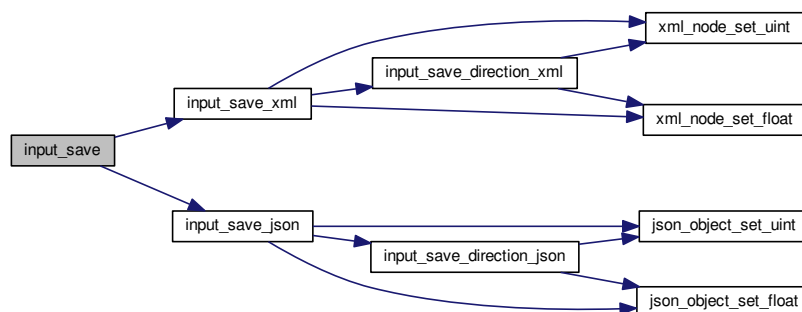
Definition at line 579 of file [interface.c](#).

```

00580 {
00581     xmlDoc *doc;
00582     JsonGenerator *generator;
00583
00584 #if DEBUG_INTERFACE
00585     fprintf (stderr, "input_save: start\n");
00586 #endif
00587
00588     // Getting the input file directory
00589     input->name = g_path_get_basename (filename);
00590     input->directory = g_path_get_dirname (filename);
00591
00592     if (input->type == INPUT_TYPE_XML)
00593     {
00594         // Opening the input file
00595         doc = xmlNewDoc ((const xmlChar *) "1.0");
00596         input_save_xml (doc);
00597
00598         // Saving the XML file
00599         xmlSaveFormatFile (filename, doc, 1);
00600
00601         // Freeing memory
00602         xmlFreeDoc (doc);
00603     }
00604     else
00605     {
00606         // Opening the input file
00607         generator = json_generator_new ();
00608         json_generator_set_pretty (generator, TRUE);
00609         input_save_json (generator);
00610
00611         // Saving the JSON file
00612         json_generator_to_file (generator, filename, NULL);
00613
00614         // Freeing memory
00615         g_object_unref (generator);
00616     }
00617
00618 #if DEBUG_INTERFACE
00619     fprintf (stderr, "input_save: end\n");
00620 #endif
00621 }

```

Here is the call graph for this function:



4.13.2.3 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

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

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

Here is the call graph for this function:



4.13.2.4 window_get_direction()

```
unsigned int window_get_direction ( )
```

Function to get the direction search method number.

Returns

Direction search method number.

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

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

Here is the call graph for this function:



4.13.2.5 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

Returns

Norm method number.

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

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

Here is the call graph for this function:



4.13.2.6 window_new()

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

Function to open the main window.

Parameters

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

Definition at line 2108 of file [interface.c](#).

```
02109 {
02110     unsigned int i;
02111     char *buffer, *buffer2, buffer3[64];
02112     char *label_algorithm[NALGORITHMS] = {
02113         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114     };
02115     char *tip_algorithm[NALGORITHMS] = {
02116         _("Monte-Carlo brute force algorithm"),
02117         _("Sweep brute force algorithm"),
02118         _("Genetic algorithm")
02119     };
02120     char *label_direction[NDIRECTIONS] = {
02121         _("_Coordinates descent"), _("_Random")
02122     };
02123     char *tip_direction[NDIRECTIONS] = {
02124         _("Coordinates direction estimate method"),
02125         _("Random direction estimate method")
02126     };
02127     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128     char *tip_norm[NNORMS] = {
02129         _("Euclidean error norm (L2)"),
02130         _("Maximum error norm (L)"),
02131         _("P error norm (Lp)"),
02132         _("Taxicab error norm (L1)")
02133     };
02134
02135     #if DEBUG_INTERFACE
02136         fprintf (stderr, "window_new: start\n");
02137     #endif
02138
02139     // Creating the window
02140     window->window = main_window
02141         = (GtkWindow *) gtk_application_window_new (application);
02142
02143     // Finish when closing the window
02144     g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148     // Setting the window title
02149     gtk_window_set_title (window->window, "MPCOTool");
02150
02151     // Creating the open button
02152     window->button_open = (GtkToolButton *) gtk_tool_button_new
02153         (gtk_image_new_from_icon_name ("document-open",
02154                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157     // Creating the save button
02158     window->button_save = (GtkToolButton *) gtk_tool_button_new
02159         (gtk_image_new_from_icon_name ("document-save",
02160                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161     g_signal_connect (window->button_save, "clicked", (void (*)(
02162         window_save,
02163         NULL);
02164
02165     // Creating the run button
02166     window->button_run = (GtkToolButton *) gtk_tool_button_new
02167         (gtk_image_new_from_icon_name ("system-run",
02168                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02169     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02170
02171     // Creating the options button
02172     window->button_options = (GtkToolButton *) gtk_tool_button_new
```

```

02172     (gtk_image_new_from_icon_name ("preferences-system",
02173                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02174     _("Options"));
02175     g_signal_connect (window->button_options, "clicked",
02176                       options_new, NULL);
02177
02178     // Creating the help button
02179     window->button_help = (GtkToolButton *) gtk_tool_button_new
02180     (gtk_image_new_from_icon_name ("help-browser",
02181                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02182     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02183
02184     // Creating the about button
02185     window->button_about = (GtkToolButton *) gtk_tool_button_new
02186     (gtk_image_new_from_icon_name ("help-about",
02187                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02188     g_signal_connect (window->button_about, "clicked",
02189                       window_about, NULL);
02190
02191     // Creating the exit button
02192     window->button_exit = (GtkToolButton *) gtk_tool_button_new
02193     (gtk_image_new_from_icon_name ("application-exit",
02194                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02195     g_signal_connect_swapped (window->button_exit, "clicked",
02196                               G_CALLBACK (g_application_quit),
02197                               G_APPLICATION (application));
02198
02199     // Creating the buttons bar
02200     window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02201     gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->
02203     button_open), 0);
02204     gtk_toolbar_insert
02205     (window->bar_buttons, GTK_TOOL_ITEM (window->
02206     button_save), 1);
02207     gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->
02209     button_run), 2);
02210     gtk_toolbar_insert
02211     (window->bar_buttons, GTK_TOOL_ITEM (window->
02212     button_options), 3);
02213     gtk_toolbar_insert
02214     (window->bar_buttons, GTK_TOOL_ITEM (window->
02215     button_help), 4);
02216     gtk_toolbar_insert
02217     (window->bar_buttons, GTK_TOOL_ITEM (window->
02218     button_about), 5);
02219     gtk_toolbar_insert
02220     (window->bar_buttons, GTK_TOOL_ITEM (window->
02221     button_exit), 6);
02222     gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02223
02224     // Creating the simulator program label and entry
02225     window->label_simulator
02226     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02227     window->button_simulator = (GtkFileChooserButton *)
02228     gtk_file_chooser_button_new (_("Simulator program"),
02229                                 GTK_FILE_CHOOSER_ACTION_OPEN);
02230     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02231                                 _("Simulator program executable file"));
02232     gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02233
02234     // Creating the evaluator program label and entry
02235     window->check_evaluator = (GtkCheckButton *)
02236     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02237     g_signal_connect (window->check_evaluator, "toggled",
02238                       window_update, NULL);
02239     window->button_evaluator = (GtkFileChooserButton *)
02240     gtk_file_chooser_button_new (_("Evaluator program"),
02241                                 GTK_FILE_CHOOSER_ACTION_OPEN);
02242     gtk_widget_set_tooltip_text
02243     (GTK_WIDGET (window->button_evaluator),
02244     _("Optional evaluator program executable file"));
02245
02246     // Creating the results files labels and entries
02247     window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02248     window->entry_result = (GtkEntry *) gtk_entry_new ();
02249     gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->entry_result), _("Best results file"));
02251     window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02252     window->entry_variables = (GtkEntry *) gtk_entry_new ();
02253     gtk_widget_set_tooltip_text
02254     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02255
02256     // Creating the files grid and attaching widgets
02257     window->grid_files = (GtkGrid *) gtk_grid_new ();
02258     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->

```



```

    label_simulator),
02249         0, 0, 1, 1);
02250     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02251         1, 0, 1, 1);
02252     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02253         0, 1, 1, 1);
02254     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02255         1, 1, 1, 1);
02256     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02257         0, 2, 1, 1);
02258     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02259         1, 2, 1, 1);
02260     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02261         0, 3, 1, 1);
02262     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02263         1, 3, 1, 1);
02264
02265     // Creating the algorithm properties
02266     window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268     window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270     gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272     _("Number of simulations to perform for each iteration"));
02273     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274     window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276     window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278     gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280     g_signal_connect
02281     (window->spin_iterations, "value-changed",
window_update, NULL);
02282     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284     window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286     gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288     _("Tolerance to set the variable interval on the next iteration"));
02289     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290     window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292     gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294     _("Number of best simulations used to set the variable interval "
02295     "on the next iteration"));
02296     window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298     window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300     gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302     _("Number of population for the genetic algorithm"));
02303     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304     window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306     window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308     gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310     _("Number of generations for the genetic algorithm"));
02311     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312     window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314     gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316     _("Ratio of mutation for the genetic algorithm"));
02317     window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319     window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321     gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323     _("Ratio of reproduction for the genetic algorithm"));
02324     window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326     window->spin_adaptation

```

```

02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328     gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332     window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335     gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338     window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                       GTK_WIDGET (window->spin_threshold));
02342     // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343     // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344     //                          GTK_ALIGN_FILL);
02345
02346     // Creating the direction search method properties
02347     window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02349     g_signal_connect (window->check_direction, "clicked",
02350                      window_update, NULL);
02351     window->grid_direction = (GtkGrid *) gtk_grid_new ();
02352     window->button_direction[0] = (GtkRadioButton *)
02353     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02354     gtk_grid_attach (window->grid_direction,
02355                     GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02356     g_signal_connect (window->button_direction[0], "clicked",
02357                      window_update, NULL);
02358     for (i = 0; ++i < NDIRECTIONS;)
02359     {
02360         window->button_direction[i] = (GtkRadioButton *)
02361         gtk_radio_button_new_with_mnemonic
02362         (gtk_radio_button_get_group (window->button_direction[0]),
02363          label_direction[i]);
02364         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02365                                     tip_direction[i]);
02366         gtk_grid_attach (window->grid_direction,
02367                         GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02368         g_signal_connect (window->button_direction[i], "clicked",
02369                          window_update, NULL);
02370     }
02371     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02372     window->spin_steps = (GtkSpinButton *)
02373     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02374     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02375     window->label_estimates
02376     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02377     window->spin_estimates = (GtkSpinButton *)
02378     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02379     window->label_relaxation
02380     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02381     window->spin_relaxation = (GtkSpinButton *)
02382     gtk_spin_button_new_with_range (0., 2., 0.001);
02383     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02384     window->label_steps),
02385                     0, NDIRECTIONS, 1, 1);
02386     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02387     window->spin_steps),
02388                     1, NDIRECTIONS, 1, 1);
02389     gtk_grid_attach (window->grid_direction,
02390                     GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02391                     1, 1);
02392     gtk_grid_attach (window->grid_direction,
02393                     GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02394                     1);
02395     gtk_grid_attach (window->grid_direction,
02396                     GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02397                     1, 1);
02398     gtk_grid_attach (window->grid_direction,
02399                     GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02400                     1, 1);
02401
02402     // Creating the array of algorithms
02403     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02404     window->button_algorithm[0] = (GtkRadioButton *)
02405     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02406     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02407                                 tip_algorithm[0]);
02408     gtk_grid_attach (window->grid_algorithm,
02409                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02410     g_signal_connect (window->button_algorithm[0], "clicked",
02411                      window_set_algorithm, NULL);
02412     for (i = 0; ++i < NALGORITHMS;)

```

```

02410     {
02411         window->button_algorithm[i] = (GtkRadioButton *)
02412             gtk_radio_button_new_with_mnemonic
02413             (gtk_radio_button_get_group (window->button_algorithm[0]),
02414              label_algorithm[i]);
02415         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416                                     tip_algorithm[i]);
02417         gtk_grid_attach (window->grid_algorithm,
02418                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419         g_signal_connect (window->button_algorithm[i], "clicked",
02420                          window_set_algorithm, NULL);
02421     }
02422     gtk_grid_attach (window->grid_algorithm,
02423                     GTK_WIDGET (window->label_simulations), 0,
02424                     NALGORITHMS, 1, 1);
02425     gtk_grid_attach (window->grid_algorithm,
02426                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427                     1);
02428     gtk_grid_attach (window->grid_algorithm,
02429                     GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430                     1, 1);
02431     gtk_grid_attach (window->grid_algorithm,
02432                     GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433                     1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435                     GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436                     1, 1);
02437     gtk_grid_attach (window->grid_algorithm,
02438                     GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439                     1);
02440     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02441 window->label_bests),
02442                     0, NALGORITHMS + 3, 1, 1);
02443     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02444 window->spin_bests), 1,
02445                     NALGORITHMS + 3, 1, 1);
02446     gtk_grid_attach (window->grid_algorithm,
02447                     GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02448                     1, 1);
02449     gtk_grid_attach (window->grid_algorithm,
02450                     GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02451                     1, 1);
02452     gtk_grid_attach (window->grid_algorithm,
02453                     GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02454                     1, 1);
02455     gtk_grid_attach (window->grid_algorithm,
02456                     GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02457                     1, 1);
02458     gtk_grid_attach (window->grid_algorithm,
02459                     GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02460                     1);
02461     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02462 window->spin_mutation),
02463                     1, NALGORITHMS + 6, 1, 1);
02464     gtk_grid_attach (window->grid_algorithm,
02465                     GTK_WIDGET (window->label_reproduction), 0,
02466                     NALGORITHMS + 7, 1, 1);
02467     gtk_grid_attach (window->grid_algorithm,
02468                     GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02469                     1, 1);
02470     gtk_grid_attach (window->grid_algorithm,
02471                     GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02472                     1, 1);
02473     gtk_grid_attach (window->grid_algorithm,
02474                     GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02475                     1, 1);
02476     gtk_grid_attach (window->grid_algorithm,
02477                     GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02478                     2, 1);
02479     gtk_grid_attach (window->grid_algorithm,
02480                     GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02481                     2, 1);
02482     gtk_grid_attach (window->grid_algorithm,
02483                     GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02484                     1, 1);
02485     gtk_grid_attach (window->grid_algorithm,
02486                     GTK_WIDGET (window->scrolled_threshold), 1,
02487                     NALGORITHMS + 11, 1, 1);
02488     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02489     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02490                       GTK_WIDGET (window->grid_algorithm));
02491     // Creating the variable widgets
02492     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02493     gtk_widget_set_tooltip_text
02494         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02495     window->id_variable = g_signal_connect

```

```

02494     (window->combo_variable, "changed", window_set_variable, NULL);
02495     window->button_add_variable
02496     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497                                                    GTK_ICON_SIZE_BUTTON);
02498     g_signal_connect
02499     (window->button_add_variable, "clicked",
02500      window_add_variable, NULL);
02501     gtk_widget_set_tooltip_text
02502     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02503     window->button_remove_variable
02504     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02505                                                    GTK_ICON_SIZE_BUTTON);
02506     g_signal_connect
02507     (window->button_remove_variable, "clicked",
02508      window_remove_variable, NULL);
02509     gtk_widget_set_tooltip_text
02510     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02511     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02512     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02513     gtk_widget_set_tooltip_text
02514     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02515     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02516     window->id_variable_label = g_signal_connect
02517     (window->entry_variable, "changed",
02518      window_label_variable, NULL);
02519     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02520     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02521     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02522     gtk_widget_set_tooltip_text
02523     (GTK_WIDGET (window->spin_min),
02524      _("Minimum initial value of the variable"));
02525     window->scrolled_min
02526     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02527     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02528                        GTK_WIDGET (window->spin_min));
02529     g_signal_connect (window->spin_min, "value-changed",
02530                      window_rangemin_variable, NULL);
02531     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02532     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02533     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02534     gtk_widget_set_tooltip_text
02535     (GTK_WIDGET (window->spin_max),
02536      _("Maximum initial value of the variable"));
02537     window->scrolled_max
02538     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02539     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02540                        GTK_WIDGET (window->spin_max));
02541     g_signal_connect (window->spin_max, "value-changed",
02542                      window_rangemax_variable, NULL);
02543     window->check_minabs = (GtkCheckButton *)
02544     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02545     g_signal_connect (window->check_minabs, "toggled",
02546                      window_update, NULL);
02547     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02548     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02549     gtk_widget_set_tooltip_text
02550     (GTK_WIDGET (window->spin_minabs),
02551      _("Minimum allowed value of the variable"));
02552     window->scrolled_minabs
02553     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02554     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02555                        GTK_WIDGET (window->spin_minabs));
02556     g_signal_connect (window->spin_minabs, "value-changed",
02557                      window_rangeminabs_variable, NULL);
02558     window->check_maxabs = (GtkCheckButton *)
02559     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02560     g_signal_connect (window->check_maxabs, "toggled",
02561                      window_update, NULL);
02562     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02563     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02564     gtk_widget_set_tooltip_text
02565     (GTK_WIDGET (window->spin_maxabs),
02566      _("Maximum allowed value of the variable"));
02567     window->scrolled_maxabs
02568     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02569     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02570                        GTK_WIDGET (window->spin_maxabs));
02571     g_signal_connect (window->spin_maxabs, "value-changed",
02572                      window_rangemaxabs_variable, NULL);
02573     window->label_precision
02574     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02575     window->spin_precision = (GtkSpinButton *)
02576     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02577     gtk_widget_set_tooltip_text
02578     (GTK_WIDGET (window->spin_precision),
02579      _("Number of precision floating point digits\n"
02580        "0 is for integer numbers"));

```

```

02576 g_signal_connect (window->spin_precision, "value-changed",
02577                  window_precision_variable, NULL);
02578 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579 window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582                             _("Number of steps sweeping the variable"));
02583 g_signal_connect (window->spin_sweeps, "value-changed",
02584                  window_update_variable, NULL);
02585 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586 window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588 gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590      _("Number of bits to encode the variable"));
02591 g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL)
02593 ;
02594 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02595 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02596     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02597 gtk_widget_set_tooltip_text
02598     (GTK_WIDGET (window->spin_step),
02599      _("Initial step size for the direction search method"));
02599 window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602                  GTK_WIDGET (window->spin_step));
02603 g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606 gtk_grid_attach (window->grid_variable,
02607                 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608 gtk_grid_attach (window->grid_variable,
02609                 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610 gtk_grid_attach (window->grid_variable,
02611                 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613                 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614 gtk_grid_attach (window->grid_variable,
02615                 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616 gtk_grid_attach (window->grid_variable,
02617                 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618 gtk_grid_attach (window->grid_variable,
02619                 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620 gtk_grid_attach (window->grid_variable,
02621                 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622 gtk_grid_attach (window->grid_variable,
02623                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624 gtk_grid_attach (window->grid_variable,
02625                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626 gtk_grid_attach (window->grid_variable,
02627                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628 gtk_grid_attach (window->grid_variable,
02629                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630 gtk_grid_attach (window->grid_variable,
02631                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632 gtk_grid_attach (window->grid_variable,
02633                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634 gtk_grid_attach (window->grid_variable,
02635                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636 gtk_grid_attach (window->grid_variable,
02637                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638 gtk_grid_attach (window->grid_variable,
02639                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640 gtk_grid_attach (window->grid_variable,
02641                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642 gtk_grid_attach (window->grid_variable,
02643                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644 gtk_grid_attach (window->grid_variable,
02645                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646 gtk_grid_attach (window->grid_variable,
02647                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                  GTK_WIDGET (window->grid_variable));
02651
02652 // Creating the experiment widgets
02653 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                             _("Experiment selector"));
02656 window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed",
02658      window_set_experiment, NULL);
02659 window->button_add_experiment
02660     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02661                                                  GTK_ICON_SIZE_BUTTON);

```

```

02661 g_signal_connect
02662 (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02663 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664 _("Add experiment"));
02665 window->button_remove_experiment
02666 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667 GTK_ICON_SIZE_BUTTON);
02668 g_signal_connect (window->button_remove_experiment, "clicked",
02669 window_remove_experiment, NULL);
02670 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
button_remove_experiment),
02671 _("Remove experiment"));
02672 window->label_experiment
02673 = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674 window->button_experiment = (GtkFileChooserButton *)
02675 gtk_file_chooser_button_new (_("Experimental data file"),
02676 GTK_FILE_CHOOSER_ACTION_OPEN);
02677 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678 _("Experimental data file"));
02679 window->id_experiment_name
02680 = g_signal_connect (window->button_experiment, "selection-changed",
02681 window_name_experiment, NULL);
02682 gtk_widget_set_hexand (GTK_WIDGET (window->button_experiment), TRUE);
02683 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684 window->spin_weight
02685 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02686 gtk_widget_set_tooltip_text
02687 (GTK_WIDGET (window->spin_weight),
02688 _("Weight factor to build the objective function"));
02689 g_signal_connect
02690 (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
02691 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692 gtk_grid_attach (window->grid_experiment,
02693 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694 gtk_grid_attach (window->grid_experiment,
02695 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696 gtk_grid_attach (window->grid_experiment,
02697 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
;
02698 gtk_grid_attach (window->grid_experiment,
02699 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700 gtk_grid_attach (window->grid_experiment,
02701 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702 gtk_grid_attach (window->grid_experiment,
02703 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704 gtk_grid_attach (window->grid_experiment,
02705 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706 for (i = 0; i < MAX_NINPUS; ++i)
02707 {
02708 snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709 window->check_template[i] = (GtkCheckButton *)
02710 gtk_check_button_new_with_label (buffer3);
02711 window->id_template[i]
02712 = g_signal_connect (window->check_template[i], "toggled",
02713 window_inputs_experiment, NULL);
02714 gtk_grid_attach (window->grid_experiment,
02715 GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716 1);
02717 window->button_template[i] =
02718 (GtkFileChooserButton *)
02719 gtk_file_chooser_button_new (_("Input template"),
02720 GTK_FILE_CHOOSER_ACTION_OPEN);
02721 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722 _("Experimental input template file"));
02723 window->id_input[i] =
02724 g_signal_connect_swapped (window->button_template[i],
02725 "selection-changed",
02726 (void (*)(void *)) window_template_experiment,
02727 (void *) (size_t) i);
02728 gtk_grid_attach (window->grid_experiment,
02729 GTK_WIDGET (window->button_template[i]),
02730 1, 3 + i, 3, 1);
02731 }
02732 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734 GTK_WIDGET (window->grid_experiment));
02735
02736 // Creating the error norm widgets
02737 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740 GTK_WIDGET (window->grid_norm));
02741 window->button_norm[0] = (GtkRadioButton *)
02742 gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),

```

```

02744         tip_norm[0]);
02745     gtk_grid_attach (window->grid_norm,
02746                     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747     g_signal_connect (window->button_norm[0], "clicked",
02748                       window_update, NULL);
02749     for (i = 0; ++i < NNORMS;)
02750     {
02751         window->button_norm[i] = (GtkRadioButton *)
02752             gtk_radio_button_new_with_mnemonic
02753             (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02754         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02755                                     tip_norm[i]);
02756         gtk_grid_attach (window->grid_norm,
02757                         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02758         g_signal_connect (window->button_norm[i], "clicked",
02759                           window_update,
02760                           NULL);
02761     }
02762     window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02763     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02764 label_p), 1, 1, 1,
02765 1);
02766     window->spin_p =
02767         (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02768                                                         G_MAXDOUBLE, 0.01);
02769     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02770                                 _("P parameter for the P error norm"));
02771     window->scrolled_p =
02772         (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02773     gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02774                       GTK_WIDGET (window->spin_p));
02775     gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02776     gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02777     gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02778 scrolled_p),
02779 1, 2, 1, 2);
02780
02781     // Creating the grid and attaching the widgets to the grid
02782     window->grid = (GtkGrid *) gtk_grid_new ();
02783     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02784 1);
02785     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02786     gtk_grid_attach (window->grid,
02787                     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02788     gtk_grid_attach (window->grid,
02789                     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02790     gtk_grid_attach (window->grid,
02791                     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02792     gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02793     gtk_container_add (GTK_CONTAINER (window->window),
02794                       GTK_WIDGET (window->grid));
02795
02796     // Setting the window logo
02797     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02798     gtk_window_set_icon (window->window, window->logo);
02799
02800     // Showing the window
02801     gtk_widget_show_all (GTK_WIDGET (window->window));
02802
02803     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02804     #if GTK_MINOR_VERSION >= 16
02805     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02806     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02807     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02808     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02809     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02810     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02811     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02812 40);
02813     #endif
02814
02815     // Reading initial example
02816     input_new ();
02817     buffer2 = g_get_current_dir ();
02818     buffer =
02819         g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02820     g_free (buffer2);
02821     window_read (buffer);
02822     g_free (buffer);
02823
02824     #if DEBUG_INTERFACE
02825     fprintf (stderr, "window_new: start\n");
02826     #endif
02827 }

```


4.13.2.7 window_read()

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

Function to read the input data of a file.

Parameters

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

Returns

1 on succes, 0 on error.

Definition at line 1904 of file [interface.c](#).

```
01905 {
01906     unsigned int i;
01907     char *buffer;
01908     #if DEBUG_INTERFACE
01909     fprintf (stderr, "window_read: start\n");
01910     #endif
01911
01912     // Reading new input file
01913     input_free ();
01914     if (!input_open (filename))
01915     {
01916         #if DEBUG_INTERFACE
01917         fprintf (stderr, "window_read: end\n");
01918         #endif
01919         return 0;
01920     }
01921
01922     // Setting GTK+ widgets data
01923     gtk_entry_set_text (window->entry_result, input->result);
01924     gtk_entry_set_text (window->entry_variables, input->
variables);
01925     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01926     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01927     g_free (buffer);
01928     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01929
01930     if (input->evaluator)
01931     {
01932         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01933         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01934         g_free (buffer);
01935     }
01936     gtk_toggle_button_set_active
01937     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01938     switch (input->algorithm)
01939     {
01940     case ALGORITHM_MONTE_CARLO:
01941         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01942     case ALGORITHM_SWEEP:
01943         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01944         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01945         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01946         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_direction),
input->nsteps);
01947         if (input->nsteps)
01948         {
01949             gtk_toggle_button_set_active
```

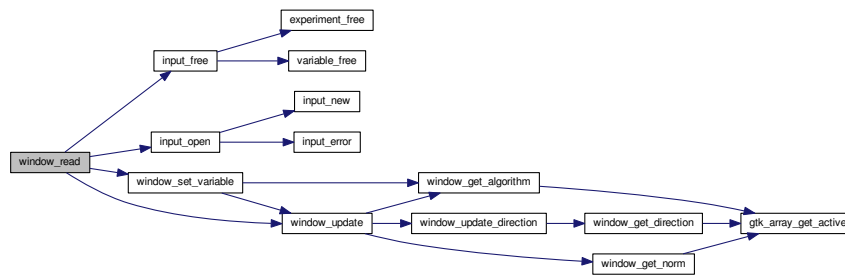


```

01955         (GTK_TOGGLE_BUTTON (window->button_direction
01956                             [input->direction]), TRUE);
01957     gtk_spin_button_set_value (window->spin_steps,
01958                               (gdouble) input->nsteps);
01959     gtk_spin_button_set_value (window->spin_relaxation,
01960                               (gdouble) input->relaxation);
01961     switch (input->direction)
01962     {
01963     case DIRECTION_METHOD_RANDOM:
01964         gtk_spin_button_set_value (window->spin_estimates,
01965                                   (gdouble) input->nestimates);
01966     }
01967     }
01968     break;
01969 default:
01970     gtk_spin_button_set_value (window->spin_population,
01971                               (gdouble) input->nsimulations);
01972     gtk_spin_button_set_value (window->spin_generations,
01973                               (gdouble) input->niterations);
01974     gtk_spin_button_set_value (window->spin_mutation,
01975                               input->mutation_ratio);
01976     gtk_spin_button_set_value (window->spin_reproduction,
01977                               input->reproduction_ratio);
01978     gtk_spin_button_set_value (window->spin_adaptation,
01979                               input->adaptation_ratio);
01980     }
01981     gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983     gtk_spin_button_set_value (window->spin_p, input->p);
01984     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01985     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01986     g_signal_handler_block (window->button_experiment,
01987                             window->id_experiment_name);
01988     gtk_combo_box_text_remove_all (window->combo_experiment);
01989     for (i = 0; i < input->nexperiments; ++i)
01990         gtk_combo_box_text_append_text (window->combo_experiment,
01991                                         input->experiment[i].name);
01992     g_signal_handler_unblock
01993     (window->button_experiment, window->
id_experiment_name);
01994     g_signal_handler_unblock (window->combo_experiment,
01995                             window->id_experiment);
01996     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01997     g_signal_handler_block (window->combo_variable, window->
id_variable);
01998     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01999     gtk_combo_box_text_remove_all (window->combo_variable);
02000     for (i = 0; i < input->nvariables; ++i)
02001         gtk_combo_box_text_append_text (window->combo_variable,
02002                                         input->variable[i].name);
02002     g_signal_handler_unblock (window->entry_variable,
02003                             window->id_variable_label);
02004     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
02005     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02006     window_set_variable ();
02007     window_update ();
02008
02009 #if DEBUG_INTERFACE
02010     fprintf (stderr, "window_read: end\n");
02011 #endif
02012     return 1;
02013 }

```

Here is the call graph for this function:



4.13.2.8 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 825 of file [interface.c](#).

```

00826 {
00827     GtkFileChooserDialog *dlg;
00828     GtkFileFilter *filter1, *filter2;
00829     char *buffer;
00830
00831     #if DEBUG_INTERFACE
00832     fprintf (stderr, "window_save: start\n");
00833     #endif
00834
00835     // Opening the saving dialog
00836     dlg = (GtkFileChooserDialog *)
00837         gtk_file_chooser_dialog_new (_("Save file"),
00838                                     window->window,
00839                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                     _("_Cancel"),
00841                                     GTK_RESPONSE_CANCEL,
00842                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00843     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844                                                    TRUE);
00845     buffer = g_build_filename (input->directory, input->name, NULL);
00846     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847     g_free (buffer);
00848
00849     // Adding XML filter
00850     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter1, "XML");
00852     gtk_file_filter_add_pattern (filter1, "*.xml");
00853     gtk_file_filter_add_pattern (filter1, "*.XML");
00854     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856     // Adding JSON filter
00857     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858     gtk_file_filter_set_name (filter2, "JSON");
00859     gtk_file_filter_add_pattern (filter2, "*.json");
00860     gtk_file_filter_add_pattern (filter2, "*.JSON");
00861     gtk_file_filter_add_pattern (filter2, "*.js");
00862     gtk_file_filter_add_pattern (filter2, "*.JS");
00863     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);

```

```

00864
00865     if (input->type == INPUT_TYPE_XML)
00866         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867     else
00868         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870     // If OK response then saving
00871     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872     {
00873         // Setting input file type
00874         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875         buffer = (char *) gtk_file_filter_get_name (filter1);
00876         if (!strcmp (buffer, "XML"))
00877             input->type = INPUT_TYPE_XML;
00878         else
00879             input->type = INPUT_TYPE_JSON;
00880
00881         // Adding properties to the root XML node
00882         input->simulator = gtk_file_chooser_get_filename
00883             (GTK_FILE_CHOOSER (window->button_simulator));
00884         if (gtk_toggle_button_get_active
00885             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886             input->evaluator = gtk_file_chooser_get_filename
00887                 (GTK_FILE_CHOOSER (window->button_evaluator));
00888         else
00889             input->evaluator = NULL;
00890         if (input->type == INPUT_TYPE_XML)
00891         {
00892             input->result
00893                 = (char *) xmlStrdup ((const xmlChar *)
00894                                         gtk_entry_get_text (window->entry_result));
00895             input->variables
00896                 = (char *) xmlStrdup ((const xmlChar *)
00897                                         gtk_entry_get_text
00898                                             (window->entry_variables));
00899         }
00900         else
00901         {
00902             input->result =
00903                 g_strdup (gtk_entry_get_text (window->entry_result));
00904             input->variables =
00905                 g_strdup (gtk_entry_get_text (window->entry_variables));
00906         }
00907
00908         // Setting the algorithm
00909         switch (window_get_algorithm ())
00910         {
00911             case ALGORITHM_MONTE_CARLO:
00912                 input->algorithm = ALGORITHM_MONTE_CARLO;
00913                 input->nsimulations
00914                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915                 input->niterations
00916                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917                 input->tolerance =
00918                     gtk_spin_button_get_value (window->spin_tolerance);
00919                 input->nbest =
00920                     gtk_spin_button_get_value_as_int (window->spin_bests);
00921                 window_save_direction ();
00922                 break;
00923             case ALGORITHM_SWEEP:
00924                 input->algorithm = ALGORITHM_SWEEP;
00925                 input->niterations
00926                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927                 input->tolerance =
00928                     gtk_spin_button_get_value (window->spin_tolerance);
00929                 input->nbest =
00930                     gtk_spin_button_get_value_as_int (window->spin_bests);
00931                 window_save_direction ();
00932                 break;
00933             default:
00934                 input->algorithm = ALGORITHM_GENETIC;
00935                 input->nsimulations
00936                     = gtk_spin_button_get_value_as_int (window->spin_population);
00937                 input->niterations
00938                     = gtk_spin_button_get_value_as_int (window->spin_generations);
00939                 input->mutation_ratio
00940                     = gtk_spin_button_get_value (window->spin_mutation);
00941                 input->reproduction_ratio
00942                     = gtk_spin_button_get_value (window->spin_reproduction);
00943                 input->adaptation_ratio
00944                     = gtk_spin_button_get_value (window->spin_adaptation);
00945                 break;
00946         }
00947         input->norm = window_get_norm ();
00948         input->p = gtk_spin_button_get_value (window->spin_p);
00949         input->threshold = gtk_spin_button_get_value (window->
spin_threshold);

```

```

00950
00951     // Saving the XML file
00952     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953     input_save (buffer);
00954
00955     // Closing and freeing memory
00956     g_free (buffer);
00957     gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959     fprintf (stderr, "window_save: end\n");
00960 #endif
00961     return 1;
00962 }
00963
00964 // Closing and freeing memory
00965 gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967     fprintf (stderr, "window_save: end\n");
00968 #endif
00969     return 0;
00970 }

```

4.13.2.9 window_template_experiment()

```

void window_template_experiment (
    void * data )

```

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

Parameters

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

Definition at line 1542 of file [interface.c](#).

```

01543 {
01544     unsigned int i, j;
01545     char *buffer;
01546     GFile *file1, *file2;
01547 #if DEBUG_INTERFACE
01548     fprintf (stderr, "window_template_experiment: start\n");
01549 #endif
01550     i = (size_t) data;
01551     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552     file1
01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555     file2 = g_file_new_for_path (input->directory);
01556     buffer = g_file_get_relative_path (file2, file1);
01557     if (input->type == INPUT_TYPE_XML)
01558         input->experiment[j].template[i] =
01559             (char *) xmlStrdup ((xmlChar *) buffer);
01560     else
01561         input->experiment[j].template[i] = g_strdup (buffer);
01562     g_free (buffer);
01563     g_object_unref (file2);
01564     g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566     fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }

```

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 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INTERFACE__H
00033 #define INTERFACE__H 1
00034
00035 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00036
00037 typedef struct
00038 {
00039     GtkWidget *dialog;
00040     GtkWidget *grid;
00041     GtkWidget *label_seed;
00042     GtkWidget *spin_seed;
00043     GtkWidget *label_threads;
00044     GtkWidget *spin_threads;
00045     GtkWidget *label_direction;
00046     GtkWidget *spin_direction;
00047 } Options;
00048
00049 typedef struct
00050 {
00051     GtkWidget *dialog;
00052     GtkWidget *label;
00053     GtkWidget *spinner;
00054     GtkWidget *grid;
00055 } Running;
00056
00057 typedef struct
00058 {
00059     GtkWidget *window;
00060     GtkWidget *grid;
00061     GtkWidget *bar_buttons;
00062     GtkWidget *button_open;
00063     GtkWidget *button_save;
00064     GtkWidget *button_run;
00065     GtkWidget *button_options;
00066     GtkWidget *button_help;
00067     GtkWidget *button_about;
00068     GtkWidget *button_exit;
00069     GtkWidget *grid_files;
00070     GtkWidget *label_simulator;
00071     GtkWidget *FileChooserButton *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *FileChooserButton *button_evaluator;
00074     GtkWidget *label_result;
00075     GtkWidget *entry_result;
00076     GtkWidget *label_variables;
00077     GtkWidget *entry_variables;
00078     GtkWidget *frame_norm;
00079     GtkWidget *grid_norm;
00080     GtkWidget *radioButton *button_norm[NNORMS];
00081     GtkWidget *label_p;
00082     GtkWidget *spin_p;
00083     GtkWidget *scrolled_p;
00084     GtkWidget *frame_algorithm;
00085     GtkWidget *grid_algorithm;
00086     GtkWidget *radioButton *button_algorithm[NALGORITHMS];
00087     GtkWidget *label_simulations;
00088     GtkWidget *spin_simulations;

```

```

00117   GtkWidget *label_iterations;
00118   GtkSpinButton *spin_iterations;
00120   GtkWidget *label_tolerance;
00121   GtkSpinButton *spin_tolerance;
00122   GtkWidget *label_bests;
00123   GtkSpinButton *spin_bests;
00124   GtkWidget *label_population;
00125   GtkSpinButton *spin_population;
00127   GtkWidget *label_generations;
00128   GtkSpinButton *spin_generations;
00130   GtkWidget *label_mutation;
00131   GtkSpinButton *spin_mutation;
00132   GtkWidget *label_reproduction;
00133   GtkSpinButton *spin_reproduction;
00135   GtkWidget *label_adaptation;
00136   GtkSpinButton *spin_adaptation;
00138   GtkCheckButton *check_direction;
00140   GtkWidget *grid_direction;
00142   GtkRadioButton *button_direction[NDIRECTIONS];
00144   GtkWidget *label_steps;
00145   GtkSpinButton *spin_steps;
00146   GtkWidget *label_estimates;
00147   GtkSpinButton *spin_estimates;
00149   GtkWidget *label_relaxation;
00151   GtkSpinButton *spin_relaxation;
00153   GtkWidget *label_threshold;
00154   GtkSpinButton *spin_threshold;
00155   GtkScrolledWindow *scrolled_threshold;
00157   GtkFrame *frame_variable;
00158   GtkWidget *grid_variable;
00159   GtkComboBoxText *combo_variable;
00161   GtkButton *button_add_variable;
00162   GtkButton *button_remove_variable;
00163   GtkWidget *label_variable;
00164   GtkEntry *entry_variable;
00165   GtkWidget *label_min;
00166   GtkSpinButton *spin_min;
00167   GtkScrolledWindow *scrolled_min;
00168   GtkWidget *label_max;
00169   GtkSpinButton *spin_max;
00170   GtkScrolledWindow *scrolled_max;
00171   GtkCheckButton *check_minabs;
00172   GtkSpinButton *spin_minabs;
00173   GtkScrolledWindow *scrolled_minabs;
00174   GtkCheckButton *check_maxabs;
00175   GtkSpinButton *spin_maxabs;
00176   GtkScrolledWindow *scrolled_maxabs;
00177   GtkWidget *label_precision;
00178   GtkSpinButton *spin_precision;
00179   GtkWidget *label_sweeps;
00180   GtkSpinButton *spin_sweeps;
00181   GtkWidget *label_bits;
00182   GtkSpinButton *spin_bits;
00183   GtkWidget *label_step;
00184   GtkSpinButton *spin_step;
00185   GtkScrolledWindow *scrolled_step;
00186   GtkFrame *frame_experiment;
00187   GtkWidget *grid_experiment;
00188   GtkComboBoxText *combo_experiment;
00189   GtkButton *button_add_experiment;
00190   GtkButton *button_remove_experiment;
00191   GtkWidget *label_experiment;
00192   GtkFileChooserButton *button_experiment;
00194   GtkWidget *label_weight;
00195   GtkSpinButton *spin_weight;
00196   GtkCheckButton *check_template[MAX_NINPUTS];
00198   GtkFileChooserButton *button_template[MAX_NINPUTS];
00200   GdkPixbuf *logo;
00201   Experiment *experiment;
00202   Variable *variable;
00203   char *application_directory;
00204   gulong id_experiment;
00205   gulong id_experiment_name;
00206   gulong id_variable;
00207   gulong id_variable_label;
00208   gulong id_template[MAX_NINPUTS];
00210   gulong id_input[MAX_NINPUTS];
00212   unsigned int nexperiments;
00213   unsigned int nvariables;
00214 } Window;
00215
00216 // Global variables
00217 extern const char *logo[];
00218 extern Options options[1];
00219 extern Running running[1];
00220 extern Window window[1];
00221

```

```

00222 // Inline functions
00223 #if GTK_MINOR_VERSION < 10
00224 static inline GtkWidget *
00225 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00226 {
00227     GtkWidget *button;
00228     GtkWidget *image;
00229     button = (GtkWidget *) gtk_button_new ();
00230     image = (GtkWidget *) gtk_image_new_from_icon_name (name, size);
00231     gtk_button_set_image (button, GTK_WIDGET (image));
00232     return button;
00233 }
00234 #endif
00235
00236 // Public functions
00237 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00238 void input_save (char *filename);
00239 void options_new ();
00240 void running_new ();
00241 unsigned int window_get_algorithm ();
00242 unsigned int window_get_direction ();
00243 unsigned int window_get_norm ();
00244 void window_save_direction ();
00245 int window_save ();
00246 void window_run ();
00247 void window_help ();
00248 void window_update_direction ();
00249 void window_update ();
00250 void window_set_algorithm ();
00251 void window_set_experiment ();
00252 void window_remove_experiment ();
00253 void window_add_experiment ();
00254 void window_name_experiment ();
00255 void window_weight_experiment ();
00256 void window_inputs_experiment ();
00257 void window_template_experiment (void *data);
00258 void window_set_variable ();
00259 void window_remove_variable ();
00260 void window_add_variable ();
00261 void window_label_variable ();
00262 void window_precision_variable ();
00263 void window_rangemin_variable ();
00264 void window_rangemax_variable ();
00265 void window_rangeminabs_variable ();
00266 void window_rangemaxabs_variable ();
00267 void window_update_variable ();
00268 int window_read (char *filename);
00269 void window_open ();
00270 void window_new (GtkApplication * application);
00271
00272 #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"

```



```

00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #if HAVE_GTK
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
00053 #endif
00054 #include "genetic/genetic.h"
00055 #include "utils.h"
00056 #include "experiment.h"
00057 #include "variable.h"
00058 #include "input.h"
00059 #include "optimize.h"
00060 #if HAVE_GTK
00061 #include "interface.h"
00062 #endif
00063
00064 #define DEBUG_MAIN 0
00065
00066 #if EXTERNAL_LIBRARY
00067 int
00068 mpcotool (int argn, char **argc)
00069 #else
00070 int
00071 main (int argn, char **argc)
00072 #endif
00073 {
00074     #if HAVE_GTK
00075     GtkApplication *application;
00076     char *buffer;
00077     #endif
00078
00079     // Starting pseudo-random numbers generator
00080     #if DEBUG_MAIN
00081     fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00082     #endif
00083     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00084
00085     // Allowing spaces in the XML data file
00086     #if DEBUG_MAIN
00087     fprintf (stderr, "main: allowing spaces in the XML data file\n");
00088     #endif
00089     xmlKeepBlanksDefault (0);
00090
00091     // Starting MPI
00092     #if HAVE_MPI
00093     #if DEBUG_MAIN
00094     fprintf (stderr, "main: starting MPI\n");
00095     #endif
00096     MPI_Init (&argn, &argc);
00097     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00098     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00099     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00100     #else
00101     ntasks = 1;
00102     #endif
00103
00104     // Resetting result and variables file names
00105     #if DEBUG_MAIN

```

```

00121     fprintf (stderr, "main: resetting result and variables file names\n");
00122 #endif
00123     input->result = input->variables = NULL;
00124
00125 #if HAVE_GTK
00126
00127     // Getting threads number and pseudo-random numbers generator seed
00128     nthreads_direction = nthreads = cores_number ();
00129     optimize->seed = DEFAULT_RANDOM_SEED;
00130
00131     // Setting local language and international floating point numbers notation
00132     setlocale (LC_ALL, "");
00133     setlocale (LC_NUMERIC, "C");
00134     window->application_directory = g_get_current_dir ();
00135     buffer = g_build_filename (window->application_directory,
00136                               LOCALE_DIR, NULL);
00137     bindtextdomain (PROGRAM_INTERFACE, buffer);
00138     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00139     textdomain (PROGRAM_INTERFACE);
00140
00141     // Initing GTK+
00142 #if !EXTERNAL_LIBRARY
00143     show_pending = process_pending;
00144 #endif
00145     gtk_disable_setlocale ();
00146     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00147                                       G_APPLICATION_FLAGS_NONE);
00148     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00149
00150     // Opening the main window
00151     g_application_run (G_APPLICATION (application), 0, NULL);
00152
00153     // Freeing memory
00154     input_free ();
00155     g_free (buffer);
00156     gtk_widget_destroy (GTK_WIDGET (window->window));
00157     g_object_unref (application);
00158     g_free (window->application_directory);
00159 #else
00160
00161     // Checking syntax
00162     if (argn < 2)
00163     {
00164         printf ("The syntax is:\n"
00165                "  ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00166                "[variables_file]\n");
00167         return 1;
00168     }
00169
00170     // Getting threads number and pseudo-random numbers generator seed
00171 #if DEBUG_MAIN
00172     fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00173             "generator seed\n");
00174 #endif
00175     nthreads_direction = nthreads = cores_number ();
00176     optimize->seed = DEFAULT_RANDOM_SEED;
00177     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00178     {
00179         nthreads_direction = nthreads = atoi (argc[2]);
00180         if (!nthreads)
00181         {
00182             printf ("Bad threads number\n");
00183             return 2;
00184         }
00185         argc += 2;
00186         argn -= 2;
00187         if (argn > 2 && !strcmp (argc[1], "-seed"))
00188         {
00189             optimize->seed = atoi (argc[2]);
00190             argc += 2;
00191             argn -= 2;
00192         }
00193     }
00194     else if (argn > 2 && !strcmp (argc[1], "-seed"))
00195     {
00196         optimize->seed = atoi (argc[2]);
00197         argc += 2;
00198         argn -= 2;
00199         if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00200         {
00201             nthreads_direction = nthreads = atoi (argc[2]);
00202             if (!nthreads)
00203             {
00204                 printf ("Bad threads number\n");
00205                 return 2;
00206             }
00207         }
00208     }

```

```

00207         argc += 2;
00208         argn -= 2;
00209     }
00210 }
00211 printf ("nthreads=%u\n", nthreads);
00212 printf ("seed=%lu\n", optimize->seed);
00213
00214 // Checking arguments
00215 #if DEBUG_MAIN
00216 fprintf (stderr, "main: checking arguments\n");
00217 #endif
00218 if (argn > 4 || argn < 2)
00219 {
00220     printf ("The syntax is:\n"
00221            " ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00222            "[variables_file]\n");
00223     return 1;
00224 }
00225 if (argn > 2)
00226     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00227 if (argn == 4)
00228     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00229
00230 // Making optimization
00231 #if DEBUG_MAIN
00232 fprintf (stderr, "main: making optimization\n");
00233 #endif
00234 if (input_open (argc[1]))
00235     optimize_open ();
00236
00237 // Freeing memory
00238 #if DEBUG_MAIN
00239 fprintf (stderr, "main: freeing memory and closing\n");
00240 #endif
00241 optimize_free ();
00242
00243 #endif
00244
00245 // Closing MPI
00246 #if HAVE_MPI
00247 MPI_Finalize ();
00248 #endif
00249
00250 // Freeing memory
00251 gsl_rng_free (optimize->rng);
00252
00253 // Closing
00254 return 0;
00255 }

```

4.17 optimize.c File Reference

Source file to define the optimization functions.

```

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

```

```
#include "input.h"
#include "optimize.h"
Include dependency graph for optimize.c:
```



Macros

- `#define DEBUG_OPTIMIZE 0`
Macro to debug optimize functions.
- `#define RM "rm"`
Macro to define the shell remove command.

Functions

- void `optimize_input` (unsigned int simulation, char *`input`, GMappedFile *`template`)
Function to write the simulation input file.
- double `optimize_parse` (unsigned int simulation, unsigned int experiment)
Function to parse input files, simulating and calculating the \ objective function.
- double `optimize_norm_euclidian` (unsigned int simulation)
Function to calculate the Euclidian error norm.
- double `optimize_norm_maximum` (unsigned int simulation)
Function to calculate the maximum error norm.
- double `optimize_norm_p` (unsigned int simulation)
Function to calculate the P error norm.
- double `optimize_norm_taxicab` (unsigned int simulation)
Function to calculate the taxicab error norm.
- void `optimize_print` ()
Function to print the results.
- void `optimize_save_variables` (unsigned int simulation, double error)
Function to save in a file the variables and the error.
- void `optimize_best` (unsigned int simulation, double value)
Function to save the best simulations.
- void `optimize_sequential` ()
Function to optimize sequentially.
- void * `optimize_thread` (ParallelData *`data`)
Function to optimize on a thread.
- void `optimize_merge` (unsigned int nsaveds, unsigned int *`simulation_best`, double *`error_best`)
Function to merge the 2 optimization results.
- void `optimize_synchronise` ()
Function to synchronise the optimization results of MPI tasks.
- void `optimize_sweep` ()
Function to optimize with the sweep algorithm.
- void `optimize_MonteCarlo` ()
Function to optimize with the Monte-Carlo algorithm.
- void `optimize_best_direction` (unsigned int simulation, double value)
Function to save the best simulation in a direction search method.

- void [optimize_direction_sequential](#) (unsigned int simulation)
Function to estimate the direction search sequentially.
- void * [optimize_direction_thread](#) ([ParallelData](#) *data)
Function to estimate the direction search on a thread.
- double [optimize_estimate_direction_random](#) (unsigned int variable, unsigned int estimate)
Function to estimate a component of the direction search vector.
- double [optimize_estimate_direction_coordinates](#) (unsigned int variable, unsigned int estimate)
Function to estimate a component of the direction search vector.
- void [optimize_step_direction](#) (unsigned int simulation)
Function to do a step of the direction search method.
- void [optimize_direction](#) ()
Function to optimize with a direction search method.
- double [optimize_genetic_objective](#) (**Entity** *entity)
Function to calculate the objective function of an entity.
- void [optimize_genetic](#) ()
Function to optimize with the genetic algorithm.
- void [optimize_save_old](#) ()
Function to save the best results on iterative methods.
- void [optimize_merge_old](#) ()
Function to merge the best results with the previous step best results on iterative methods.
- void [optimize_refine](#) ()
Function to refine the search ranges of the variables in iterative algorithms.
- void [optimize_step](#) ()
Function to do a step of the iterative algorithm.
- void [optimize_iterate](#) ()
Function to iterate the algorithm.
- void [optimize_free](#) ()
Function to free the memory used by the [Optimize](#) struct.
- void [optimize_open](#) ()
Function to open and perform a optimization.

Variables

- int [ntasks](#)
Number of tasks.
- unsigned int [nthreads](#)
Number of threads.
- unsigned int [nthreads_direction](#)
Number of threads for the direction search method.
- GMutex [mutex](#) [1]
Mutex struct.
- void(* [optimize_algorithm](#))()
Pointer to the function to perform a optimization algorithm step.
- double(* [optimize_estimate_direction](#))(unsigned int variable, unsigned int estimate)
Pointer to the function to estimate the direction.
- double(* [optimize_norm](#))(unsigned int simulation)
Pointer to the error norm function.
- [Optimize](#) [optimize](#) [1]
Optimization data.

4.17.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.17.2 Function Documentation

4.17.2.1 optimize_best()

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

Function to save the best simulations.

Parameters

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

Definition at line [468](#) of file [optimize.c](#).

```
00469 {
00470     unsigned int i, j;
00471     double e;
00472     #if DEBUG_OPTIMIZE
00473     fprintf (stderr, "optimize_best: start\n");
00474     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475             optimize->nsaveds, optimize->nbest);
00476     #endif
00477     if (optimize->nsaveds < optimize->nbest
00478         || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480         if (optimize->nsaveds < optimize->nbest)
00481             ++optimize->nsaveds;
00482         optimize->error_best[optimize->nsaveds - 1] = value;
00483         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484         for (i = optimize->nsaveds; --i;)
00485         {
00486             if (optimize->error_best[i] < optimize->
00487                 error_best[i - 1])
00488             {
00489                 j = optimize->simulation_best[i];
00490                 e = optimize->error_best[i];
00491                 optimize->simulation_best[i] = optimize->
00492                     simulation_best[i - 1];
```

```

00491         optimize->error_best[i] = optimize->
error_best[i - 1];
00492         optimize->simulation_best[i - 1] = j;
00493         optimize->error_best[i - 1] = e;
00494     }
00495     else
00496         break;
00497 }
00498 }
00499 #if DEBUG_OPTIMIZE
00500 fprintf(stderr, "optimize_best: end\n");
00501 #endif
00502 }

```

4.17.2.2 optimize_best_direction()

```

void optimize_best_direction (
    unsigned int simulation,
    double value )

```

Function to save the best simulation in a direction search method.

Parameters

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

Definition at line 795 of file [optimize.c](#).

```

00796 {
00797 #if DEBUG_OPTIMIZE
00798     fprintf(stderr, "optimize_best_direction: start\n");
00799     fprintf(stderr,
00800         "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801         simulation, value, optimize->error_best[0]);
00802 #endif
00803     if (value < optimize->error_best[0])
00804     {
00805         optimize->error_best[0] = value;
00806         optimize->simulation_best[0] = simulation;
00807 #if DEBUG_OPTIMIZE
00808         fprintf(stderr,
00809             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810             simulation, value);
00811 #endif
00812     }
00813 #if DEBUG_OPTIMIZE
00814     fprintf(stderr, "optimize_best_direction: end\n");
00815 #endif
00816 }

```

4.17.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

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

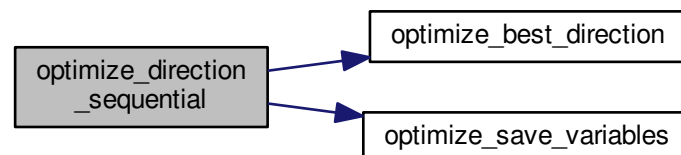
Definition at line 825 of file [optimize.c](#).

```

00826 {
00827     unsigned int i, j;
00828     double e;
00829     #if DEBUG_OPTIMIZE
00830     fprintf (stderr, "optimize_direction_sequential: start\n");
00831     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832             "nend_direction=%u\n",
00833             optimize->nstart_direction, optimize->
nend_direction);
00834     #endif
00835     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837         j = simulation + i;
00838         e = optimize_norm (j);
00839         optimize_best_direction (j, e);
00840         optimize_save_variables (j, e);
00841         if (e < optimize->threshold)
00842         {
00843             optimize->stop = 1;
00844             break;
00845         }
00846     #if DEBUG_OPTIMIZE
00847     fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00848     #endif
00849     }
00850     #if DEBUG_OPTIMIZE
00851     fprintf (stderr, "optimize_direction_sequential: end\n");
00852     #endif
00853 }

```

Here is the call graph for this function:



4.17.2.4 optimize_direction_thread()

```

void * optimize_direction_thread (
    ParallelData * data )

```

Function to estimate the direction search on a thread.

Parameters

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

Returns

NULL

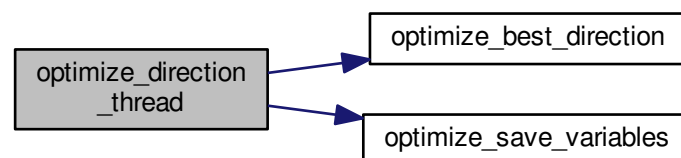
Definition at line 863 of file [optimize.c](#).

```

00864 {
00865     unsigned int i, thread;
00866     double e;
00867     #if DEBUG_OPTIMIZE
00868         fprintf (stderr, "optimize_direction_thread: start\n");
00869     #endif
00870     thread = data->thread;
00871     #if DEBUG_OPTIMIZE
00872         fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873                 thread,
00874                 optimize->thread_direction[thread],
00875                 optimize->thread_direction[thread + 1]);
00876     #endif
00877     for (i = optimize->thread_direction[thread];
00878          i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880         e = optimize_norm (i);
00881         g_mutex_lock (mutex);
00882         optimize_best_direction (i, e);
00883         optimize_save_variables (i, e);
00884         if (e < optimize->threshold)
00885             optimize->stop = 1;
00886         g_mutex_unlock (mutex);
00887         if (optimize->stop)
00888             break;
00889     #if DEBUG_OPTIMIZE
00890         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00891     #endif
00892     }
00893     #if DEBUG_OPTIMIZE
00894         fprintf (stderr, "optimize_direction_thread: end\n");
00895     #endif
00896     g_thread_exit (NULL);
00897     return NULL;
00898 }

```

Here is the call graph for this function:



4.17.2.5 optimize_estimate_direction_coordinates()

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

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 937 of file [optimize.c](#).

```

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

4.17.2.6 optimize_estimate_direction_random()

```

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

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 910 of file [optimize.c](#).

```

00912 {
00913     double x;
00914     #if DEBUG_OPTIMIZE
00915     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00916     #endif
00917     x = optimize->direction[variable]
00918         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00919         step[variable];
00920     #if DEBUG_OPTIMIZE
00921     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00922             variable, x);
00923     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00924     #endif
00925     return x;
00926 }
```

4.17.2.7 optimize_genetic_objective()

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

Function to calculate the objective function of an entity.

Parameters

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

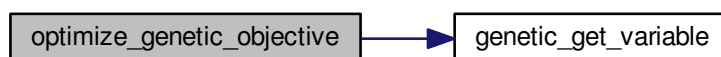
Returns

objective function value.

Definition at line 1104 of file [optimize.c](#).

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

Here is the call graph for this function:



4.17.2.8 optimize_input()

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

Function to write the simulation input file.

Parameters

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

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

```

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

```

```

00176 #endif
00177     return;
00178 }

```

4.17.2.9 optimize_merge()

```

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

```

Function to merge the 2 optimization results.

Parameters

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

Definition at line 591 of file [optimize.c](#).

```

00593 {
00594     unsigned int i, j, k, s[optimize->nbest];
00595     double e[optimize->nbest];
00596     #if DEBUG_OPTIMIZE
00597         fprintf (stderr, "optimize_merge: start\n");
00598     #endif
00599     i = j = k = 0;
00600     do
00601     {
00602         if (i == optimize->nsaveds)
00603         {
00604             s[k] = simulation_best[j];
00605             e[k] = error_best[j];
00606             ++j;
00607             ++k;
00608             if (j == nsaveds)
00609                 break;
00610         }
00611         else if (j == nsaveds)
00612         {
00613             s[k] = optimize->simulation_best[i];
00614             e[k] = optimize->error_best[i];
00615             ++i;
00616             ++k;
00617             if (i == optimize->nsaveds)
00618                 break;
00619         }
00620         else if (optimize->error_best[i] > error_best[j])
00621         {
00622             s[k] = simulation_best[j];
00623             e[k] = error_best[j];
00624             ++j;
00625             ++k;
00626         }
00627         else
00628         {
00629             s[k] = optimize->simulation_best[i];
00630             e[k] = optimize->error_best[i];
00631             ++i;
00632             ++k;
00633         }
00634     }
00635     while (k < optimize->nbest);
00636     optimize->nsaveds = k;
00637     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638     memcpy (optimize->error_best, e, k * sizeof (double));

```

```

00639 #if DEBUG_OPTIMIZE
00640     fprintf (stderr, "optimize_merge: end\n");
00641 #endif
00642 }

```

4.17.2.10 optimize_norm_euclidian()

```

double optimize_norm_euclidian (
    unsigned int simulation )

```

Function to calculate the Euclidian error norm.

Parameters

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

Returns

Euclidian error norm.

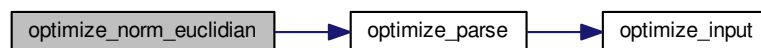
Definition at line 300 of file [optimize.c](#).

```

00301 {
00302     double e, ei;
00303     unsigned int i;
00304     #if DEBUG_OPTIMIZE
00305         fprintf (stderr, "optimize_norm_euclidian: start\n");
00306     #endif
00307     e = 0.;
00308     for (i = 0; i < optimize->nexperiments; ++i)
00309     {
00310         ei = optimize_parse (simulation, i);
00311         e += ei * ei;
00312     }
00313     e = sqrt (e);
00314     #if DEBUG_OPTIMIZE
00315         fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316         fprintf (stderr, "optimize_norm_euclidian: end\n");
00317     #endif
00318     return e;
00319 }

```

Here is the call graph for this function:



4.17.2.11 optimize_norm_maximum()

```

double optimize_norm_maximum (
    unsigned int simulation )

```

Function to calculate the maximum error norm.

Parameters

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

Returns

Maximum error norm.

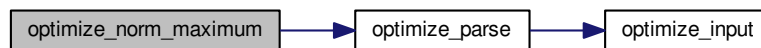
Definition at line 329 of file [optimize.c](#).

```

00330 {
00331     double e, ei;
00332     unsigned int i;
00333     #if DEBUG_OPTIMIZE
00334     fprintf (stderr, "optimize_norm_maximum: start\n");
00335     #endif
00336     e = 0.;
00337     for (i = 0; i < optimize->nexperiments; ++i)
00338     {
00339         ei = fabs (optimize_parse (simulation, i));
00340         e = fmax (e, ei);
00341     }
00342     #if DEBUG_OPTIMIZE
00343     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00344     fprintf (stderr, "optimize_norm_maximum: end\n");
00345     #endif
00346     return e;
00347 }

```

Here is the call graph for this function:



4.17.2.12 optimize_norm_p()

```

double optimize_norm_p (
    unsigned int simulation )

```

Function to calculate the P error norm.

Parameters

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

Returns

P error norm.

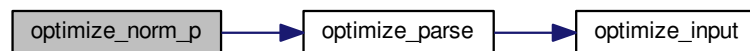
Definition at line 357 of file [optimize.c](#).

```

00358 {
00359     double e, ei;
00360     unsigned int i;
00361     #if DEBUG_OPTIMIZE
00362     fprintf (stderr, "optimize_norm_p: start\n");
00363     #endif
00364     e = 0.;
00365     for (i = 0; i < optimize->nexperiments; ++i)
00366     {
00367         ei = fabs (optimize_parse (simulation, i));
00368         e += pow (ei, optimize->p);
00369     }
00370     e = pow (e, 1. / optimize->p);
00371     #if DEBUG_OPTIMIZE
00372     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00373     fprintf (stderr, "optimize_norm_p: end\n");
00374     #endif
00375     return e;
00376 }

```

Here is the call graph for this function:



4.17.2.13 optimize_norm_taxicab()

```

double optimize_norm_taxicab (
    unsigned int simulation )

```

Function to calculate the taxicab error norm.

Parameters

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

Returns

Taxicab error norm.

Definition at line 386 of file [optimize.c](#).

```

00387 {
00388     double e;
00389     unsigned int i;
00390     #if DEBUG_OPTIMIZE
00391     fprintf (stderr, "optimize_norm_taxicab: start\n");
00392     #endif
00393     e = 0.;
00394     for (i = 0; i < optimize->nexperiments; ++i)

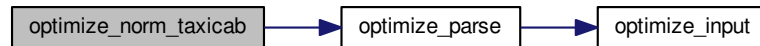
```

```

00395     e += fabs (optimize_parse (simulation, i));
00396 #if DEBUG_OPTIMIZE
00397     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398     fprintf (stderr, "optimize_norm_taxicab: end\n");
00399 #endif
00400     return e;
00401 }

```

Here is the call graph for this function:



4.17.2.14 optimize_parse()

```

double optimize_parse (
    unsigned int simulation,
    unsigned int experiment )

```

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

Parameters

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

Returns

Objective function value.

Definition at line 191 of file [optimize.c](#).

```

00192 {
00193     unsigned int i;
00194     double e;
00195     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196         *buffer3, *buffer4;
00197     FILE *file_result;
00198
00199 #if DEBUG_OPTIMIZE
00200     fprintf (stderr, "optimize_parse: start\n");
00201     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202             simulation, experiment);
00203 #endif
00204
00205     // Opening input files
00206     for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209                 experiment);
00210 #if DEBUG_OPTIMIZE
00211         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif

```

```

00213         optimize_input (simulation, &input[i][0],
00214                         optimize->file[i][experiment]);
00215     }
00216     for (; i < MAX_NINPUTS; ++i)
00217         strcpy (&input[i][0], "");
00218     #if DEBUG_OPTIMIZE
00219     fprintf (stderr, "optimize_parse: parsing end\n");
00220     #endif
00221
00222     // Performing the simulation
00223     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224     buffer2 = g_path_get_dirname (optimize->simulator);
00225     buffer3 = g_path_get_basename (optimize->simulator);
00226     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227     snprintf (buffer, 512, "%s\\%s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230     g_free (buffer4);
00231     g_free (buffer3);
00232     g_free (buffer2);
00233     #if DEBUG_OPTIMIZE
00234     fprintf (stderr, "optimize_parse: %s\n", buffer);
00235     #endif
00236     system (buffer);
00237
00238     // Checking the objective value function
00239     if (optimize->evaluator)
00240     {
00241         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242         buffer2 = g_path_get_dirname (optimize->evaluator);
00243         buffer3 = g_path_get_basename (optimize->evaluator);
00244         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245         snprintf (buffer, 512, "%s\\%s %s %s %s",
00246             buffer4, output, optimize->experiment[experiment], result);
00247         g_free (buffer4);
00248         g_free (buffer3);
00249         g_free (buffer2);
00250         #if DEBUG_OPTIMIZE
00251         fprintf (stderr, "optimize_parse: %s\n", buffer);
00252         #endif
00253         system (buffer);
00254         file_result = g_fopen (result, "r");
00255         e = atof (fgets (buffer, 512, file_result));
00256         fclose (file_result);
00257     }
00258     else
00259     {
00260         strcpy (result, "");
00261         file_result = g_fopen (output, "r");
00262         e = atof (fgets (buffer, 512, file_result));
00263         fclose (file_result);
00264     }
00265
00266     // Removing files
00267     #if !DEBUG_OPTIMIZE
00268     for (i = 0; i < optimize->ninputs; ++i)
00269     {
00270         if (optimize->file[i][0])
00271         {
00272             snprintf (buffer, 512, RM " %s", &input[i][0]);
00273             system (buffer);
00274         }
00275     }
00276     snprintf (buffer, 512, RM " %s %s", output, result);
00277     system (buffer);
00278     #endif
00279
00280     // Processing pending events
00281     if (show_pending)
00282         show_pending ();
00283
00284     #if DEBUG_OPTIMIZE
00285     fprintf (stderr, "optimize_parse: end\n");
00286     #endif
00287
00288     // Returning the objective function
00289     return e * optimize->weight[experiment];
00290 }

```

Here is the call graph for this function:



4.17.2.15 optimize_save_variables()

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

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

Parameters

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

Definition at line 439 of file [optimize.c](#).

```
00440 {
00441     unsigned int i;
00442     char buffer[64];
00443     #if DEBUG_OPTIMIZE
00444     fprintf (stderr, "optimize_save_variables: start\n");
00445     #endif
00446     for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449         fprintf (optimize->file_variables, buffer,
00450             optimize->value[simulation * optimize->
00451                 nvariables + i]);
00452     }
00453     fprintf (optimize->file_variables, "%.14le\n", error);
00454     fflush (optimize->file_variables);
00455     #if DEBUG_OPTIMIZE
00456     fprintf (stderr, "optimize_save_variables: end\n");
00457     #endif
00458 }
```

4.17.2.16 optimize_step_direction()

```
void optimize_step_direction (
    unsigned int simulation )
```

Function to do a step of the direction search method.

Parameters

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

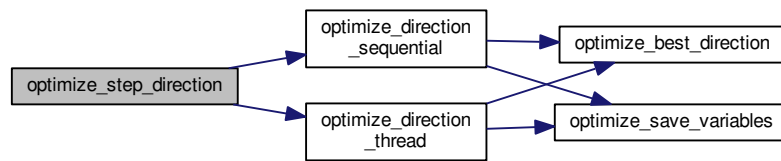
Definition at line 968 of file `optimize.c`.

```

00969 {
00970     GThread *thread[nthreads_direction];
00971     ParallelData data[nthreads_direction];
00972     unsigned int i, j, k, b;
00973     #if DEBUG_OPTIMIZE
00974     fprintf (stderr, "optimize_step_direction: start\n");
00975     #endif
00976     for (i = 0; i < optimize->nestimates; ++i)
00977     {
00978         k = (simulation + i) * optimize->nvariables;
00979         b = optimize->simulation_best[0] * optimize->
nvariables;
00980     #if DEBUG_OPTIMIZE
00981         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00982                 simulation + i, optimize->simulation_best[0]);
00983     #endif
00984         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00985         {
00986             #if DEBUG_OPTIMIZE
00987             fprintf (stderr,
00988                     "optimize_step_direction: estimate=%u best%u=%.14le\n",
00989                     i, j, optimize->value[b]);
00990             #endif
00991             optimize->value[k]
00992             = optimize->value[b] + optimize_estimate_direction (j,
i);
00993             optimize->value[k] = fmin (fmax (optimize->value[k],
00994                                             optimize->rangeminabs[j]),
00995                                       optimize->rangemaxabs[j]);
00996             #if DEBUG_OPTIMIZE
00997             fprintf (stderr,
00998                     "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00999                     i, j, optimize->value[k]);
01000             #endif
01001         }
01002     }
01003     if (nthreads_direction == 1)
01004         optimize_direction_sequential (simulation);
01005     else
01006     {
01007         for (i = 0; i <= nthreads_direction; ++i)
01008         {
01009             optimize->thread_direction[i]
01010             = simulation + optimize->nstart_direction
01011             + i * (optimize->nend_direction - optimize->
nstart_direction)
01012             / nthreads_direction;
01013             #if DEBUG_OPTIMIZE
01014             fprintf (stderr,
01015                     "optimize_step_direction: i=%u thread_direction=%u\n",
01016                     i, optimize->thread_direction[i]);
01017             #endif
01018         }
01019         for (i = 0; i < nthreads_direction; ++i)
01020         {
01021             data[i].thread = i;
01022             thread[i] = g_thread_new
01023             (NULL, (void (*)(void *)) optimize_direction_thread, &data[i]);
01024         }
01025         for (i = 0; i < nthreads_direction; ++i)
01026             g_thread_join (thread[i]);
01027     }
01028     #if DEBUG_OPTIMIZE
01029     fprintf (stderr, "optimize_step_direction: end\n");
01030     #endif
01031 }

```

Here is the call graph for this function:



4.17.2.17 optimize_thread()

```
void * optimize_thread (
    ParallelData * data )
```

Function to optimize on a thread.

Parameters

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

Returns

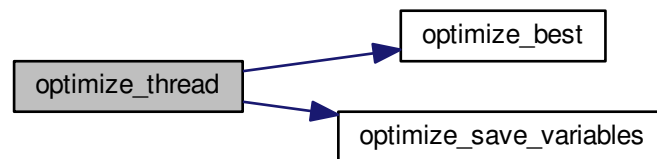
NULL

Definition at line 545 of file [optimize.c](#).

```

00546 {
00547     unsigned int i, thread;
00548     double e;
00549     #if DEBUG_OPTIMIZE
00550     fprintf (stderr, "optimize_thread: start\n");
00551     #endif
00552     thread = data->thread;
00553     #if DEBUG_OPTIMIZE
00554     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00555             optimize->thread[thread], optimize->thread[thread + 1]);
00556     #endif
00557     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00558     {
00559         e = optimize_norm (i);
00560         g_mutex_lock (mutex);
00561         optimize_best (i, e);
00562         optimize_save_variables (i, e);
00563         if (e < optimize->threshold)
00564             optimize->stop = 1;
00565         g_mutex_unlock (mutex);
00566         if (optimize->stop)
00567             break;
00568     #if DEBUG_OPTIMIZE
00569     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570     #endif
00571     }
00572     #if DEBUG_OPTIMIZE
00573     fprintf (stderr, "optimize_thread: end\n");
00574     #endif
00575     g_thread_exit (NULL);
00576     return NULL;
00577 }
```

Here is the call graph for this function:



4.18 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 2012-2016, 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 int ntasks;
00079 unsigned int nthreads;
00080 unsigned int nthreads_direction;
00082 GMutex mutex[1];
00083 void (*optimize_algorithm) ();
00085 double (*optimize_estimate_direction) (unsigned int variable,
00086                                         unsigned int estimate);
00088 double (*optimize_norm) (unsigned int simulation);
00090 Optimize optimize[1];
00091
00103 void
00104 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00105 {
00106     unsigned int i;
00107     char buffer[32], value[32], *buffer2, *buffer3, *content;
00108     FILE *file;
00109     gsize length;
00110     GRegex *regex;
00111
00112     #if DEBUG_OPTIMIZE
00113         fprintf (stderr, "optimize_input: start\n");
00114     #endif
00115
00116     // Checking the file
00117     if (!template)
00118         goto optimize_input_end;
00119
00120     // Opening template
00121     content = g_mapped_file_get_contents (template);
00122     length = g_mapped_file_get_length (template);
00123     #if DEBUG_OPTIMIZE
00124         fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length,
00125                 content);
00126     #endif
00127     file = g_fopen (input, "w");
00128
00129     // Parsing template
00130     for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132         #if DEBUG_OPTIMIZE
00133             fprintf (stderr, "optimize_input: variable=%u\n", i);
00134         #endif
00135         snprintf (buffer, 32, "@variable%u@", i + 1);
00136         regex = g_regex_new (buffer, 0, 0, NULL);
00137         if (i == 0)
00138         {
00139             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                                 optimize->label[i], 0, NULL);
00141             #if DEBUG_OPTIMIZE
00142                 fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143             #endif
00144         }
00145         else
00146         {
00147             length = strlen (buffer3);
00148             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                                 optimize->label[i], 0, NULL);
00150             g_free (buffer3);
00151         }
00152         g_regex_unref (regex);
00153         length = strlen (buffer2);
00154         snprintf (buffer, 32, "@value%u@", i + 1);
00155         regex = g_regex_new (buffer, 0, 0, NULL);
00156         snprintf (value, 32, format[optimize->precision[i]],
00157                 optimize->value[simulation * optimize->nvariables + i]);
00158
00159         #if DEBUG_OPTIMIZE
00160             fprintf (stderr, "optimize_input: value=%s\n", value);
00161         #endif
00162         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                           0, NULL);
00164         g_free (buffer2);
00165         g_regex_unref (regex);
00166     }
00167
00168     // Saving input file

```

```

00169     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170     g_free (buffer3);
00171     fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177     return;
00178 }
00179
00190 double
00191 optimize_parse (unsigned int simulation, unsigned int experiment)
00192 {
00193     unsigned int i;
00194     double e;
00195     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196         *buffer3, *buffer4;
00197     FILE *file_result;
00198
00199 #if DEBUG_OPTIMIZE
00200     fprintf (stderr, "optimize_parse: start\n");
00201     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202             simulation, experiment);
00203 #endif
00204
00205     // Opening input files
00206     for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209                 experiment);
00210 #if DEBUG_OPTIMIZE
00211         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif
00213         optimize_input (simulation, &input[i][0],
00214                         optimize->file[i][experiment]);
00215     }
00216     for (; i < MAX_NINPUTS; ++i)
00217         strcpy (&input[i][0], "");
00218 #if DEBUG_OPTIMIZE
00219     fprintf (stderr, "optimize_parse: parsing end\n");
00220 #endif
00221
00222     // Performing the simulation
00223     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224     buffer2 = g_path_get_dirname (optimize->simulator);
00225     buffer3 = g_path_get_basename (optimize->simulator);
00226     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230     g_free (buffer4);
00231     g_free (buffer3);
00232     g_free (buffer2);
00233 #if DEBUG_OPTIMIZE
00234     fprintf (stderr, "optimize_parse: %s\n", buffer);
00235 #endif
00236     system (buffer);
00237
00238     // Checking the objective value function
00239     if (optimize->evaluator)
00240     {
00241         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242         buffer2 = g_path_get_dirname (optimize->evaluator);
00243         buffer3 = g_path_get_basename (optimize->evaluator);
00244         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245         snprintf (buffer, 512, "\"%s\" %s %s %s",
00246                 buffer4, output, optimize->experiment[experiment], result);
00247         g_free (buffer4);
00248         g_free (buffer3);
00249         g_free (buffer2);
00250 #if DEBUG_OPTIMIZE
00251         fprintf (stderr, "optimize_parse: %s\n", buffer);
00252 #endif
00253         system (buffer);
00254         file_result = g_fopen (result, "r");
00255         e = atof (fgets (buffer, 512, file_result));
00256         fclose (file_result);
00257     }
00258     else
00259     {
00260         strcpy (result, "");
00261         file_result = g_fopen (output, "r");
00262         e = atof (fgets (buffer, 512, file_result));
00263         fclose (file_result);
00264     }
00265

```

```

00266 // Removing files
00267 #if !DEBUG_OPTIMIZE
00268 for (i = 0; i < optimize->ninputs; ++i)
00269 {
00270     if (optimize->file[i][0])
00271     {
00272         snprintf (buffer, 512, RM " %s", &input[i][0]);
00273         system (buffer);
00274     }
00275 }
00276 snprintf (buffer, 512, RM " %s %s", output, result);
00277 system (buffer);
00278 #endif
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 // Returning the objective function
00287 return e * optimize->weight[experiment];
00288 }
00289
00290 double
00291 optimize_norm_euclidian (unsigned int simulation)
00292 {
00293     double e, ei;
00294     unsigned int i;
00295     #if DEBUG_OPTIMIZE
00296     fprintf (stderr, "optimize_norm_euclidian: start\n");
00297     #endif
00298     e = 0.;
00299     for (i = 0; i < optimize->nexperiments; ++i)
00300     {
00301         ei = optimize_parse (simulation, i);
00302         e += ei * ei;
00303     }
00304     e = sqrt (e);
00305     #if DEBUG_OPTIMIZE
00306     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00307     fprintf (stderr, "optimize_norm_euclidian: end\n");
00308     #endif
00309     return e;
00310 }
00311
00312 double
00313 optimize_norm_maximum (unsigned int simulation)
00314 {
00315     double e, ei;
00316     unsigned int i;
00317     #if DEBUG_OPTIMIZE
00318     fprintf (stderr, "optimize_norm_maximum: start\n");
00319     #endif
00320     e = 0.;
00321     for (i = 0; i < optimize->nexperiments; ++i)
00322     {
00323         ei = fabs (optimize_parse (simulation, i));
00324         e = fmax (e, ei);
00325     }
00326     #if DEBUG_OPTIMIZE
00327     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00328     fprintf (stderr, "optimize_norm_maximum: end\n");
00329     #endif
00330     return e;
00331 }
00332
00333 double
00334 optimize_norm_p (unsigned int simulation)
00335 {
00336     double e, ei;
00337     unsigned int i;
00338     #if DEBUG_OPTIMIZE
00339     fprintf (stderr, "optimize_norm_p: start\n");
00340     #endif
00341     e = 0.;
00342     for (i = 0; i < optimize->nexperiments; ++i)
00343     {
00344         ei = fabs (optimize_parse (simulation, i));
00345         e += pow (ei, optimize->p);
00346     }
00347     e = pow (e, 1. / optimize->p);
00348     #if DEBUG_OPTIMIZE
00349     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00350     fprintf (stderr, "optimize_norm_p: end\n");
00351     #endif
00352 }

```

```

00374 #endif
00375     return e;
00376 }
00377
00385 double
00386 optimize_norm_taxicab (unsigned int simulation)
00387 {
00388     double e;
00389     unsigned int i;
00390     #if DEBUG_OPTIMIZE
00391     fprintf (stderr, "optimize_norm_taxicab: start\n");
00392     #endif
00393     e = 0.;
00394     for (i = 0; i < optimize->nexperiments; ++i)
00395         e += fabs (optimize_parse (simulation, i));
00396     #if DEBUG_OPTIMIZE
00397     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398     fprintf (stderr, "optimize_norm_taxicab: end\n");
00399     #endif
00400     return e;
00401 }
00402
00407 void
00408 optimize_print ()
00409 {
00410     unsigned int i;
00411     char buffer[512];
00412     #if HAVE_MPI
00413     if (optimize->mpi_rank)
00414         return;
00415     #endif
00416     printf ("%s\n", _("Best result"));
00417     fprintf (optimize->file_result, "%s\n", _("Best result"));
00418     printf ("error = %.15le\n", optimize->error_old[0]);
00419     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00420     for (i = 0; i < optimize->nvariables; ++i)
00421     {
00422         snprintf (buffer, 512, "%s = %s\n",
00423                 optimize->label[i], format[optimize->precision[i]]);
00424         printf (buffer, optimize->value_old[i]);
00425         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00426     }
00427     fflush (optimize->file_result);
00428 }
00429
00438 void
00439 optimize_save_variables (unsigned int simulation, double error)
00440 {
00441     unsigned int i;
00442     char buffer[64];
00443     #if DEBUG_OPTIMIZE
00444     fprintf (stderr, "optimize_save_variables: start\n");
00445     #endif
00446     for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449         fprintf (optimize->file_variables, buffer,
00450                 optimize->value[simulation * optimize->nvariables + i]);
00451     }
00452     fprintf (optimize->file_variables, "%.14le\n", error);
00453     fflush (optimize->file_variables);
00454     #if DEBUG_OPTIMIZE
00455     fprintf (stderr, "optimize_save_variables: end\n");
00456     #endif
00457 }
00458
00467 void
00468 optimize_best (unsigned int simulation, double value)
00469 {
00470     unsigned int i, j;
00471     double e;
00472     #if DEBUG_OPTIMIZE
00473     fprintf (stderr, "optimize_best: start\n");
00474     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475             optimize->nsaveds, optimize->nbest);
00476     #endif
00477     if (optimize->nsaveds < optimize->nbest
00478         || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480         if (optimize->nsaveds < optimize->nbest)
00481             ++optimize->nsaveds;
00482         optimize->error_best[optimize->nsaveds - 1] = value;
00483         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484         for (i = optimize->nsaveds; --i;)
00485         {
00486             if (optimize->error_best[i] < optimize->error_best[i - 1])

```

```

00487         {
00488             j = optimize->simulation_best[i];
00489             e = optimize->error_best[i];
00490             optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00491             optimize->error_best[i] = optimize->error_best[i - 1];
00492             optimize->simulation_best[i - 1] = j;
00493             optimize->error_best[i - 1] = e;
00494         }
00495         else
00496             break;
00497     }
00498 }
00499 #if DEBUG_OPTIMIZE
00500 fprintf (stderr, "optimize_best: end\n");
00501 #endif
00502 }
00503
00504 void
00505 optimize_sequential ()
00506 {
00507     unsigned int i;
00508     double e;
00509     #if DEBUG_OPTIMIZE
00510     fprintf (stderr, "optimize_sequential: start\n");
00511     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00512             optimize->nstart, optimize->nend);
00513     #endif
00514     for (i = optimize->nstart; i < optimize->nend; ++i)
00515     {
00516         e = optimize_norm (i);
00517         optimize_best (i, e);
00518         optimize_save_variables (i, e);
00519         if (e < optimize->threshold)
00520         {
00521             optimize->stop = 1;
00522             break;
00523         }
00524     }
00525     #if DEBUG_OPTIMIZE
00526     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00527     #endif
00528 }
00529 #if DEBUG_OPTIMIZE
00530 fprintf (stderr, "optimize_sequential: end\n");
00531 #endif
00532 }
00533
00534 void *
00535 optimize_thread (ParallelData * data)
00536 {
00537     unsigned int i, thread;
00538     double e;
00539     #if DEBUG_OPTIMIZE
00540     fprintf (stderr, "optimize_thread: start\n");
00541     #endif
00542     thread = data->thread;
00543     #if DEBUG_OPTIMIZE
00544     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00545             optimize->thread[thread], optimize->thread[thread + 1]);
00546     #endif
00547     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00548     {
00549         e = optimize_norm (i);
00550         g_mutex_lock (mutex);
00551         optimize_best (i, e);
00552         optimize_save_variables (i, e);
00553         if (e < optimize->threshold)
00554         {
00555             optimize->stop = 1;
00556             g_mutex_unlock (mutex);
00557             break;
00558         }
00559     }
00560     #if DEBUG_OPTIMIZE
00561     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00562     #endif
00563 }
00564 #if DEBUG_OPTIMIZE
00565 fprintf (stderr, "optimize_thread: end\n");
00566 #endif
00567 g_thread_exit (NULL);
00568 return NULL;
00569 }
00570
00571 void
00572 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00573                double *error_best)
00574 {
00575     unsigned int i, j, k, s[optimize->nbest];

```

```

00595     double e[optimize->nbest];
00596 #if DEBUG_OPTIMIZE
00597     fprintf (stderr, "optimize_merge: start\n");
00598 #endif
00599     i = j = k = 0;
00600     do
00601     {
00602         if (i == optimize->nsaveds)
00603         {
00604             s[k] = simulation_best[j];
00605             e[k] = error_best[j];
00606             ++j;
00607             ++k;
00608             if (j == nsaveds)
00609                 break;
00610         }
00611         else if (j == nsaveds)
00612         {
00613             s[k] = optimize->simulation_best[i];
00614             e[k] = optimize->error_best[i];
00615             ++i;
00616             ++k;
00617             if (i == optimize->nsaveds)
00618                 break;
00619         }
00620         else if (optimize->error_best[i] > error_best[j])
00621         {
00622             s[k] = simulation_best[j];
00623             e[k] = error_best[j];
00624             ++j;
00625             ++k;
00626         }
00627         else
00628         {
00629             s[k] = optimize->simulation_best[i];
00630             e[k] = optimize->error_best[i];
00631             ++i;
00632             ++k;
00633         }
00634     }
00635     while (k < optimize->nbest);
00636     optimize->nsaveds = k;
00637     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638     memcpy (optimize->error_best, e, k * sizeof (double));
00639 #if DEBUG_OPTIMIZE
00640     fprintf (stderr, "optimize_merge: end\n");
00641 #endif
00642 }
00643
00644 #if HAVE_MPI
00645 void
00646 optimize_synchronise ()
00647 {
00648     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00649     double error_best[optimize->nbest];
00650     MPI_Status mpi_stat;
00651 #if DEBUG_OPTIMIZE
00652     fprintf (stderr, "optimize_synchronise: start\n");
00653 #endif
00654     if (optimize->mpi_rank == 0)
00655     {
00656         for (i = 1; i < ntasks; ++i)
00657         {
00658             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00659             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00660                     MPI_COMM_WORLD, &mpi_stat);
00661             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00662                     MPI_COMM_WORLD, &mpi_stat);
00663             optimize_merge (nsaveds, simulation_best, error_best);
00664             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00665             if (stop)
00666                 optimize->stop = 1;
00667         }
00668         for (i = 1; i < ntasks; ++i)
00669             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00670     }
00671     else
00672     {
00673         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00674         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00675                 MPI_COMM_WORLD);
00676         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00677                 MPI_COMM_WORLD);
00678         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00679         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00680         if (stop)
00681             optimize->stop = 1;
00682     }
00683 }

```

```

00686     }
00687 #if DEBUG_OPTIMIZE
00688     fprintf (stderr, "optimize_synchronise: end\n");
00689 #endif
00690 }
00691 #endif
00692
00693 void
00694 optimize_sweep ()
00695 {
00696     unsigned int i, j, k, l;
00697     double e;
00698     GThread *thread[nthreads];
00699     ParallelData data[nthreads];
00700 #if DEBUG_OPTIMIZE
00701     fprintf (stderr, "optimize_sweep: start\n");
00702 #endif
00703     for (i = 0; i < optimize->nsimulations; ++i)
00704     {
00705         k = i;
00706         for (j = 0; j < optimize->nvariables; ++j)
00707         {
00708             l = k % optimize->nsweeps[j];
00709             k /= optimize->nsweeps[j];
00710             e = optimize->rangemin[j];
00711             if (optimize->nsweeps[j] > 1)
00712                 e += 1 * (optimize->rangemax[j] - optimize->rangemin[j])
00713                     / (optimize->nsweeps[j] - 1);
00714             optimize->value[i * optimize->nvariables + j] = e;
00715         }
00716     }
00717     optimize->nsaveds = 0;
00718     if (nthreads <= 1)
00719         optimize_sequential ();
00720     else
00721     {
00722         for (i = 0; i < nthreads; ++i)
00723         {
00724             data[i].thread = i;
00725             thread[i] =
00726                 g_thread_new (NULL, (void (*)(void *)) optimize_thread, &data[i]);
00727         }
00728         for (i = 0; i < nthreads; ++i)
00729             g_thread_join (thread[i]);
00730     }
00731 #if HAVE_MPI
00732     // Communicating tasks results
00733     optimize_synchronise ();
00734 #endif
00735 #if DEBUG_OPTIMIZE
00736     fprintf (stderr, "optimize_sweep: end\n");
00737 #endif
00738 }
00739
00740 void
00741 optimize_MonteCarlo ()
00742 {
00743     unsigned int i, j;
00744     GThread *thread[nthreads];
00745     ParallelData data[nthreads];
00746 #if DEBUG_OPTIMIZE
00747     fprintf (stderr, "optimize_MonteCarlo: start\n");
00748 #endif
00749     for (i = 0; i < optimize->nsimulations; ++i)
00750     {
00751         for (j = 0; j < optimize->nvariables; ++j)
00752             optimize->value[i * optimize->nvariables + j]
00753                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00754                   * (optimize->rangemax[j] - optimize->rangemin[j]);
00755         optimize->nsaveds = 0;
00756         if (nthreads <= 1)
00757             optimize_sequential ();
00758         else
00759         {
00760             for (i = 0; i < nthreads; ++i)
00761             {
00762                 data[i].thread = i;
00763                 thread[i] =
00764                     g_thread_new (NULL, (void (*)(void *)) optimize_thread, &data[i]);
00765             }
00766             for (i = 0; i < nthreads; ++i)
00767                 g_thread_join (thread[i]);
00768         }
00769 #if HAVE_MPI
00770     // Communicating tasks results
00771     optimize_synchronise ();
00772 #endif
00773 #if DEBUG_OPTIMIZE
00774

```

```

00781     fprintf (stderr, "optimize_MonteCarlo: end\n");
00782 #endif
00783 }
00784
00794 void
00795 optimize_best_direction (unsigned int simulation, double value)
00796 {
00797     #if DEBUG_OPTIMIZE
00798         fprintf (stderr, "optimize_best_direction: start\n");
00799         fprintf (stderr,
00800             "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801             simulation, value, optimize->error_best[0]);
00802     #endif
00803     if (value < optimize->error_best[0])
00804     {
00805         optimize->error_best[0] = value;
00806         optimize->simulation_best[0] = simulation;
00807     #if DEBUG_OPTIMIZE
00808         fprintf (stderr,
00809             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810             simulation, value);
00811     #endif
00812     }
00813 #if DEBUG_OPTIMIZE
00814     fprintf (stderr, "optimize_best_direction: end\n");
00815 #endif
00816 }
00817
00824 void
00825 optimize_direction_sequential (unsigned int simulation)
00826 {
00827     unsigned int i, j;
00828     double e;
00829     #if DEBUG_OPTIMIZE
00830         fprintf (stderr, "optimize_direction_sequential: start\n");
00831         fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832             "nend_direction=%u\n",
00833             optimize->nstart_direction, optimize->nend_direction);
00834     #endif
00835     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837         j = simulation + i;
00838         e = optimize_norm (j);
00839         optimize_best_direction (j, e);
00840         optimize_save_variables (j, e);
00841         if (e < optimize->threshold)
00842         {
00843             optimize->stop = 1;
00844             break;
00845         }
00846     #if DEBUG_OPTIMIZE
00847         fprintf (stderr, "optimize_direction_sequential: i=%u e=%.14le\n", i, e);
00848     #endif
00849     }
00850 #if DEBUG_OPTIMIZE
00851     fprintf (stderr, "optimize_direction_sequential: end\n");
00852 #endif
00853 }
00854
00862 void *
00863 optimize_direction_thread (ParallelData * data)
00864 {
00865     unsigned int i, thread;
00866     double e;
00867     #if DEBUG_OPTIMIZE
00868         fprintf (stderr, "optimize_direction_thread: start\n");
00869     #endif
00870     thread = data->thread;
00871     #if DEBUG_OPTIMIZE
00872         fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873             thread,
00874             optimize->thread_direction[thread],
00875             optimize->thread_direction[thread + 1]);
00876     #endif
00877     for (i = optimize->thread_direction[thread];
00878         i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880         e = optimize_norm (i);
00881         g_mutex_lock (mutex);
00882         optimize_best_direction (i, e);
00883         optimize_save_variables (i, e);
00884         if (e < optimize->threshold)
00885             optimize->stop = 1;
00886         g_mutex_unlock (mutex);
00887         if (optimize->stop)
00888             break;
00889     #if DEBUG_OPTIMIZE

```



```

00890     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00891 #endif
00892 }
00893 #if DEBUG_OPTIMIZE
00894 fprintf (stderr, "optimize_direction_thread: end\n");
00895 #endif
00896 g_thread_exit (NULL);
00897 return NULL;
00898 }
00899
00900 double
00910 optimize_estimate_direction_random (unsigned int variable,
00911                                     unsigned int estimate)
00912 {
00913     double x;
00914 #if DEBUG_OPTIMIZE
00915     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00916 #endif
00917     x = optimize->direction[variable]
00918         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00919 #if DEBUG_OPTIMIZE
00920     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00921             variable, x);
00922     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00923 #endif
00924     return x;
00925 }
00926
00936 double
00937 optimize_estimate_direction_coordinates (unsigned int variable,
00938                                         unsigned int estimate)
00939 {
00940     double x;
00941 #if DEBUG_OPTIMIZE
00942     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00943 #endif
00944     x = optimize->direction[variable];
00945     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00946     {
00947         if (estimate & 1)
00948             x += optimize->step[variable];
00949         else
00950             x -= optimize->step[variable];
00951     }
00952 #if DEBUG_OPTIMIZE
00953     fprintf (stderr,
00954             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00955             variable, x);
00956     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00957 #endif
00958     return x;
00959 }
00960
00967 void
00968 optimize_step_direction (unsigned int simulation)
00969 {
00970     GThread *thread[nthreads_direction];
00971     ParallelData data[nthreads_direction];
00972     unsigned int i, j, k, b;
00973 #if DEBUG_OPTIMIZE
00974     fprintf (stderr, "optimize_step_direction: start\n");
00975 #endif
00976     for (i = 0; i < optimize->nestimates; ++i)
00977     {
00978         k = (simulation + i) * optimize->nvariables;
00979         b = optimize->simulation_best[0] * optimize->nvariables;
00980 #if DEBUG_OPTIMIZE
00981         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00982                 simulation + i, optimize->simulation_best[0]);
00983 #endif
00984         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00985         {
00986 #if DEBUG_OPTIMIZE
00987             fprintf (stderr,
00988                     "optimize_step_direction: estimate=%u best%u=%.14le\n",
00989                     i, j, optimize->value[b]);
00990 #endif
00991             optimize->value[k]
00992                 = optimize->value[b] + optimize_estimate_direction (j, i);
00993             optimize->value[k] = fmin (fmax (optimize->value[k],
00994                                         optimize->rangeminabs[j]),
00995                                     optimize->rangemaxabs[j]);
00996 #if DEBUG_OPTIMIZE
00997             fprintf (stderr,
00998                     "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00999                     i, j, optimize->value[k]);
01000 #endif

```

```

01001     }
01002 }
01003 if (nthreads_direction == 1)
01004     optimize_direction_sequential (simulation);
01005 else
01006 {
01007     for (i = 0; i <= nthreads_direction; ++i)
01008     {
01009         optimize->thread_direction[i]
01010             = simulation + optimize->nstart_direction
01011             + i * (optimize->nend_direction - optimize->
nstart_direction)
01012             / nthreads_direction;
01013 #if DEBUG_OPTIMIZE
01014         fprintf (stderr,
01015             "optimize_step_direction: i=%u thread_direction=%u\n",
01016             i, optimize->thread_direction[i]);
01017 #endif
01018     }
01019     for (i = 0; i < nthreads_direction; ++i)
01020     {
01021         data[i].thread = i;
01022         thread[i] = g_thread_new
01023             (NULL, (void (*)(void *)) optimize_direction_thread, &data[i]);
01024     }
01025     for (i = 0; i < nthreads_direction; ++i)
01026         g_thread_join (thread[i]);
01027 }
01028 #if DEBUG_OPTIMIZE
01029 fprintf (stderr, "optimize_step_direction: end\n");
01030 #endif
01031 }
01032
01037 void
01038 optimize_direction ()
01039 {
01040     unsigned int i, j, k, b, s, adjust;
01041 #if DEBUG_OPTIMIZE
01042     fprintf (stderr, "optimize_direction: start\n");
01043 #endif
01044     for (i = 0; i < optimize->nvariables; ++i)
01045         optimize->direction[i] = 0.;
01046     b = optimize->simulation_best[0] * optimize->nvariables;
01047     s = optimize->nsimulations;
01048     adjust = 1;
01049     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01050     {
01051 #if DEBUG_OPTIMIZE
01052         fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01053             i, optimize->simulation_best[0]);
01054 #endif
01055         optimize_step_direction (s);
01056         k = optimize->simulation_best[0] * optimize->nvariables;
01057 #if DEBUG_OPTIMIZE
01058         fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01059             i, optimize->simulation_best[0]);
01060 #endif
01061         if (k == b)
01062         {
01063             if (adjust)
01064                 for (j = 0; j < optimize->nvariables; ++j)
01065                     optimize->step[j] *= 0.5;
01066             for (j = 0; j < optimize->nvariables; ++j)
01067                 optimize->direction[j] = 0.;
01068             adjust = 1;
01069         }
01070         else
01071         {
01072             for (j = 0; j < optimize->nvariables; ++j)
01073             {
01074 #if DEBUG_OPTIMIZE
01075                 fprintf (stderr,
01076                     "optimize_direction: best=%u old=%u\n",
01077                     j, optimize->value[k + j], j, optimize->value[b + j]);
01078 #endif
01079                 optimize->direction[j]
01080                     = (1. - optimize->relaxation) * optimize->direction[j]
01081                     + optimize->relaxation
01082                     * (optimize->value[k + j] - optimize->value[b + j]);
01083 #if DEBUG_OPTIMIZE
01084                 fprintf (stderr, "optimize_direction: direction=%u\n",
01085                     j, optimize->direction[j]);
01086 #endif
01087             }
01088             adjust = 0;
01089         }
01090     }

```

```

01091 #if DEBUG_OPTIMIZE
01092     fprintf (stderr, "optimize_direction: end\n");
01093 #endif
01094 }
01095
01103 double
01104 optimize_genetic_objective (Entity * entity)
01105 {
01106     unsigned int j;
01107     double objective;
01108     char buffer[64];
01109     #if DEBUG_OPTIMIZE
01110     fprintf (stderr, "optimize_genetic_objective: start\n");
01111     #endif
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114         optimize->value[entity->id * optimize->nvariables + j]
01115             = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117     objective = optimize_norm (entity->id);
01118     g_mutex_lock (mutex);
01119     for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122         fprintf (optimize->file_variables, buffer,
01123             genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125     fprintf (optimize->file_variables, "%.14le\n", objective);
01126     g_mutex_unlock (mutex);
01127     #if DEBUG_OPTIMIZE
01128     fprintf (stderr, "optimize_genetic_objective: end\n");
01129     #endif
01130     return objective;
01131 }
01132
01137 void
01138 optimize_genetic ()
01139 {
01140     char *best_genome;
01141     double best_objective, *best_variable;
01142     #if DEBUG_OPTIMIZE
01143     fprintf (stderr, "optimize_genetic: start\n");
01144     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01145         nthreads);
01146     fprintf (stderr,
01147         "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01148         optimize->nvariables, optimize->nsimulations,
01149         optimize->niterations);
01150     fprintf (stderr,
01151         "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01152         optimize->mutation_ratio, optimize->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
01171         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01172     optimize->error_old[0] = best_objective;
01173     memcpy (optimize->value_old, best_variable,
01174         optimize->nvariables * sizeof (double));
01175     g_free (best_genome);
01176     g_free (best_variable);
01177     optimize_print ();
01178     #if DEBUG_OPTIMIZE
01179     fprintf (stderr, "optimize_genetic: end\n");
01180     #endif
01181 }
01182
01187 void
01188 optimize_save_old ()
01189 {
01190     unsigned int i, j;
01191     #if DEBUG_OPTIMIZE
01192     fprintf (stderr, "optimize_save_old: start\n");

```

```

01193     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01194 #endif
01195     memcpy (optimize->error_old, optimize->error_best,
01196             optimize->nbest * sizeof (double));
01197     for (i = 0; i < optimize->nbest; ++i)
01198     {
01199         j = optimize->simulation_best[i];
01200 #if DEBUG_OPTIMIZE
01201         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01202 #endif
01203         memcpy (optimize->value_old + i * optimize->nvariables,
01204                 optimize->value + j * optimize->nvariables,
01205                 optimize->nvariables * sizeof (double));
01206     }
01207 #if DEBUG_OPTIMIZE
01208     for (i = 0; i < optimize->nvariables; ++i)
01209         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01210                 i, optimize->value_old[i]);
01211     fprintf (stderr, "optimize_save_old: end\n");
01212 #endif
01213 }
01214
01220 void
01221 optimize_merge_old ()
01222 {
01223     unsigned int i, j, k;
01224     double v[optimize->nbest * optimize->nvariables], e[optimize->
01225             nbest],
01226            *enew, *eold;
01227 #if DEBUG_OPTIMIZE
01228     fprintf (stderr, "optimize_merge_old: start\n");
01229 #endif
01230     enew = optimize->error_best;
01231     eold = optimize->error_old;
01232     i = j = k = 0;
01233     do
01234     {
01235         if (*enew < *eold)
01236         {
01237             memcpy (v + k * optimize->nvariables,
01238                     optimize->value
01239                     + optimize->simulation_best[i] * optimize->
01240                     nvariables,
01241                     optimize->nvariables * sizeof (double));
01242             e[k] = *enew;
01243             ++k;
01244             ++enew;
01245             ++i;
01246         }
01247         else
01248         {
01249             memcpy (v + k * optimize->nvariables,
01250                     optimize->value_old + j * optimize->nvariables,
01251                     optimize->nvariables * sizeof (double));
01252             e[k] = *eold;
01253             ++k;
01254             ++eold;
01255             ++j;
01256         }
01257     } while (k < optimize->nbest);
01258     memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01259     memcpy (optimize->error_old, e, k * sizeof (double));
01260 #if DEBUG_OPTIMIZE
01261     fprintf (stderr, "optimize_merge_old: end\n");
01262 #endif
01263 }
01264
01269 void
01270 optimize_refine ()
01271 {
01272     unsigned int i, j;
01273     double d;
01274 #if HAVE_MPI
01275     MPI_Status mpi_stat;
01276 #endif
01277 #if DEBUG_OPTIMIZE
01278     fprintf (stderr, "optimize_refine: start\n");
01279 #endif
01280 #if HAVE_MPI
01281     if (!optimize->mpi_rank)
01282     {
01283 #endif
01284         for (j = 0; j < optimize->nvariables; ++j)
01285         {
01286             optimize->rangemin[j] = optimize->rangemax[j]
01287                 = optimize->value_old[j];

```

```

01288     }
01289     for (i = 0; ++i < optimize->nbest;)
01290     {
01291         for (j = 0; j < optimize->nvariables; ++j)
01292         {
01293             optimize->rangemin[j]
01294             = fmin (optimize->rangemin[j],
01295                     optimize->value_old[i * optimize->nvariables + j]);
01296             optimize->rangemax[j]
01297             = fmax (optimize->rangemax[j],
01298                     optimize->value_old[i * optimize->nvariables + j]);
01299         }
01300     }
01301     for (j = 0; j < optimize->nvariables; ++j)
01302     {
01303         d = optimize->tolerance
01304         * (optimize->rangemax[j] - optimize->rangemin[j]);
01305         switch (optimize->algorithm)
01306         {
01307             case ALGORITHM_MONTE_CARLO:
01308                 d *= 0.5;
01309                 break;
01310             default:
01311                 if (optimize->nsweeps[j] > 1)
01312                     d /= optimize->nsweeps[j] - 1;
01313                 else
01314                     d = 0.;
01315         }
01316         optimize->rangemin[j] -= d;
01317         optimize->rangemin[j]
01318         = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01319         optimize->rangemax[j] += d;
01320         optimize->rangemax[j]
01321         = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01322         printf ("%s min=%lg max=%lg\n", optimize->label[j],
01323                 optimize->rangemin[j], optimize->rangemax[j]);
01324         fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01325                 optimize->label[j], optimize->rangemin[j],
01326                 optimize->rangemax[j]);
01327     }
01328     #if HAVE_MPI
01329     for (i = 1; i < ntasks; ++i)
01330     {
01331         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01332                  1, MPI_COMM_WORLD);
01333         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01334                  1, MPI_COMM_WORLD);
01335     }
01336     }
01337     else
01338     {
01339         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01340                  MPI_COMM_WORLD, &mpi_stat);
01341         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01342                  MPI_COMM_WORLD, &mpi_stat);
01343     }
01344     #endif
01345     #if DEBUG_OPTIMIZE
01346     fprintf (stderr, "optimize_refine: end\n");
01347     #endif
01348 }
01349
01350 void
01351 optimize_step ()
01352 {
01353     #if DEBUG_OPTIMIZE
01354     fprintf (stderr, "optimize_step: start\n");
01355     #endif
01356     optimize_algorithm ();
01357     if (optimize->nsteps)
01358         optimize_direction ();
01359     #if DEBUG_OPTIMIZE
01360     fprintf (stderr, "optimize_step: end\n");
01361     #endif
01362 }
01363
01364 void
01365 optimize_iterate ()
01366 {
01367     unsigned int i;
01368     #if DEBUG_OPTIMIZE
01369     fprintf (stderr, "optimize_iterate: start\n");
01370     #endif
01371     optimize->error_old =
01372     (double *) g_malloc (optimize->nbest * sizeof (double));
01373     optimize->value_old =
01374     (double *) g_malloc (optimize->nbest * optimize->nvariables *

```

```

01383             sizeof (double));
01384 optimize_step ();
01385 optimize_save_old ();
01386 optimize_refine ();
01387 optimize_print ();
01388 for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01389 {
01390     optimize_step ();
01391     optimize_merge_old ();
01392     optimize_refine ();
01393     optimize_print ();
01394 }
01395 #if DEBUG_OPTIMIZE
01396 fprintf (stderr, "optimize_iterate: end\n");
01397 #endif
01398 }
01399
01400 void
01401 optimize_free ()
01402 {
01403     unsigned int i, j;
01404     #if DEBUG_OPTIMIZE
01405     fprintf (stderr, "optimize_free: start\n");
01406     #endif
01407     for (j = 0; j < optimize->ninputs; ++j)
01408     {
01409         for (i = 0; i < optimize->nexperiments; ++i)
01410             g_mapped_file_unref (optimize->file[j][i]);
01411         g_free (optimize->file[j]);
01412     }
01413     g_free (optimize->error_old);
01414     g_free (optimize->value_old);
01415     g_free (optimize->value);
01416     g_free (optimize->genetic_variable);
01417     #if DEBUG_OPTIMIZE
01418     fprintf (stderr, "optimize_free: end\n");
01419     #endif
01420 }
01421
01422 void
01423 optimize_open ()
01424 {
01425     GTimeZone *tz;
01426     GDateTime *t0, *t;
01427     unsigned int i, j;
01428     #if DEBUG_OPTIMIZE
01429     char *buffer;
01430     fprintf (stderr, "optimize_open: start\n");
01431     #endif
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     // Obtaining and initing the pseudo-random numbers generator seed
01439     #if DEBUG_OPTIMIZE
01440     fprintf (stderr, "optimize_open: getting initial seed\n");
01441     #endif
01442     if (optimize->seed == DEFAULT_RANDOM_SEED)
01443         optimize->seed = input->seed;
01444     gsl_rng_set (optimize->rng, optimize->seed);
01445     // Replacing the working directory
01446     #if DEBUG_OPTIMIZE
01447     fprintf (stderr, "optimize_open: replacing the working directory\n");
01448     #endif
01449     g_chdir (input->directory);
01450     // Getting results file names
01451     optimize->result = input->result;
01452     optimize->variables = input->variables;
01453     // Obtaining the simulator file
01454     optimize->simulator = input->simulator;
01455     // Obtaining the evaluator file
01456     optimize->evaluator = input->evaluator;
01457     // Reading the algorithm
01458     optimize->algorithm = input->algorithm;
01459     switch (optimize->algorithm)
01460     {
01461     case ALGORITHM_MONTE_CARLO:

```

```

01478     optimize_algorithm = optimize_MonteCarlo;
01479     break;
01480 case ALGORITHM_SWEEP:
01481     optimize_algorithm = optimize_sweep;
01482     break;
01483 default:
01484     optimize_algorithm = optimize_genetic;
01485     optimize->mutation_ratio = input->mutation_ratio;
01486     optimize->reproduction_ratio = input->
reproduction_ratio;
01487     optimize->adaptation_ratio = input->adaptation_ratio;
01488 }
01489 optimize->nvariables = input->nvariables;
01490 optimize->nsimulations = input->nsimulations;
01491 optimize->niterations = input->niterations;
01492 optimize->nbest = input->nbest;
01493 optimize->tolerance = input->tolerance;
01494 optimize->nsteps = input->nsteps;
01495 optimize->nestimates = 0;
01496 optimize->threshold = input->threshold;
01497 optimize->stop = 0;
01498 if (input->nsteps)
01499 {
01500     optimize->relaxation = input->relaxation;
01501     switch (input->direction)
01502     {
01503     case DIRECTION_METHOD_COORDINATES:
01504         optimize->nestimates = 2 * optimize->nvariables;
01505         optimize_estimate_direction =
01506             optimize_estimate_direction_coordinates;
01507         break;
01508     default:
01509         optimize->nestimates = input->nestimates;
01510         optimize_estimate_direction =
01511             optimize_estimate_direction_random;
01512     }
01513 }
01514 #if DEBUG_OPTIMIZE
01515 fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01516 #endif
01517 optimize->simulation_best
01518     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01519 optimize->error_best =
01520     (double *) alloca (optimize->nbest * sizeof (double));
01521
01522 // Reading the experimental data
01523 #if DEBUG_OPTIMIZE
01524 buffer = g_get_current_dir ();
01525 fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01526 g_free (buffer);
01527 #endif
01528 optimize->nexperiments = input->nexperiments;
01529 optimize->ninputs = input->experiment->ninputs;
01530 optimize->experiment
01531     = (char **) alloca (input->nexperiments * sizeof (char *));
01532 optimize->weight =
01533     (double *) alloca (input->nexperiments * sizeof (double));
01534 for (i = 0; i < input->experiment->ninputs; ++i)
01535     optimize->file[i] = (GMappedFile **)
01536         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01537 for (i = 0; i < input->nexperiments; ++i)
01538 {
01539 #if DEBUG_OPTIMIZE
01540     fprintf (stderr, "optimize_open: i=%u\n", i);
01541 #endif
01542     optimize->experiment[i] = input->experiment[i].
name;
01543     optimize->weight[i] = input->experiment[i].weight;
01544 #if DEBUG_OPTIMIZE
01545     fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01546         optimize->experiment[i], optimize->weight[i]);
01547 #endif
01548     for (j = 0; j < input->experiment->ninputs; ++j)
01549     {
01550 #if DEBUG_OPTIMIZE
01551         fprintf (stderr, "optimize_open: template%u\n", j + 1);
01552 #endif
01553         optimize->file[j][i]
01554             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01555     }
01556 }
01557
01558 // Reading the variables data
01559 #if DEBUG_OPTIMIZE
01560 fprintf (stderr, "optimize_open: reading variables\n");
01561 #endif

```

```

01562 optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01563 j = input->nvariables * sizeof (double);
01564 optimize->rangemin = (double *) alloca (j);
01565 optimize->rangeminabs = (double *) alloca (j);
01566 optimize->rangemax = (double *) alloca (j);
01567 optimize->rangemaxabs = (double *) alloca (j);
01568 optimize->step = (double *) alloca (j);
01569 j = input->nvariables * sizeof (unsigned int);
01570 optimize->precision = (unsigned int *) alloca (j);
01571 optimize->nsweeps = (unsigned int *) alloca (j);
01572 optimize->nbits = (unsigned int *) alloca (j);
01573 for (i = 0; i < input->nvariables; ++i)
01574 {
01575     optimize->label[i] = input->variable[i].name;
01576     optimize->rangemin[i] = input->variable[i].rangemin;
01577     optimize->rangeminabs[i] = input->variable[i].
rangeminabs;
01578     optimize->rangemax[i] = input->variable[i].rangemax;
01579     optimize->rangemaxabs[i] = input->variable[i].
rangemaxabs;
01580     optimize->precision[i] = input->variable[i].
precision;
01581     optimize->step[i] = input->variable[i].step;
01582     optimize->nsweeps[i] = input->variable[i].nsweeps;
01583     optimize->nbits[i] = input->variable[i].nbits;
01584 }
01585 if (input->algorithm == ALGORITHM_SWEEP)
01586 {
01587     optimize->nsimulations = 1;
01588     for (i = 0; i < input->nvariables; ++i)
01589     {
01590         if (input->algorithm == ALGORITHM_SWEEP)
01591         {
01592             optimize->nsimulations *= optimize->nsweeps[i];
01593 #if DEBUG_OPTIMIZE
01594             fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01595                     optimize->nsweeps[i], optimize->nsimulations);
01596 #endif
01597         }
01598     }
01599 }
01600 if (optimize->nsteps)
01601     optimize->direction
01602     = (double *) alloca (optimize->nvariables * sizeof (double));
01603
01604 // Setting error norm
01605 switch (input->norm)
01606 {
01607     case ERROR_NORM_EUCLIDIAN:
01608         optimize_norm = optimize_norm_euclidian;
01609         break;
01610     case ERROR_NORM_MAXIMUM:
01611         optimize_norm = optimize_norm_maximum;
01612         break;
01613     case ERROR_NORM_P:
01614         optimize_norm = optimize_norm_p;
01615         optimize->p = input->p;
01616         break;
01617     default:
01618         optimize_norm = optimize_norm_taxicab;
01619 }
01620
01621 // Allocating values
01622 #if DEBUG_OPTIMIZE
01623     fprintf (stderr, "optimize_open: allocating variables\n");
01624     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01625             optimize->nvariables, optimize->algorithm);
01626 #endif
01627 optimize->genetic_variable = NULL;
01628 if (optimize->algorithm == ALGORITHM_GENETIC)
01629 {
01630     optimize->genetic_variable = (GeneticVariable *)
01631     g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01632     for (i = 0; i < optimize->nvariables; ++i)
01633     {
01634 #if DEBUG_OPTIMIZE
01635         fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01636                 i, optimize->rangemin[i], optimize->rangemax[i],
01637                 optimize->nbits[i]);
01638 #endif
01639         optimize->genetic_variable[i].minimum = optimize->
rangemin[i];
01640         optimize->genetic_variable[i].maximum = optimize->
rangemax[i];
01641         optimize->genetic_variable[i].nbits = optimize->nbits[i];
01642     }
01643 }

```



```

01644 #if DEBUG_OPTIMIZE
01645     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01646             optimize->nvariables, optimize->nsimulations);
01647 #endif
01648     optimize->value = (double *)
01649         g_malloc ((optimize->nsimulations
01650                 + optimize->nestimates * optimize->nsteps)
01651                 * optimize->nvariables * sizeof (double));
01652
01653     // Calculating simulations to perform for each task
01654     #if HAVE_MPI
01655     #if DEBUG_OPTIMIZE
01656     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01657             optimize->mpi_rank, ntasks);
01658     #endif
01659     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01660         ntasks;
01661     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01662         ntasks;
01663     if (optimize->nsteps)
01664     {
01665         optimize->nstart_direction
01666             = optimize->mpi_rank * optimize->nestimates / ntasks;
01667         optimize->nend_direction
01668             = (1 + optimize->mpi_rank) * optimize->nestimates /
01669                 ntasks;
01670     }
01671 #else
01672     optimize->nstart = 0;
01673     optimize->nend = optimize->nsimulations;
01674     if (optimize->nsteps)
01675     {
01676         optimize->nstart_direction = 0;
01677         optimize->nend_direction = optimize->nestimates;
01678     }
01679 #endif
01680 #if DEBUG_OPTIMIZE
01681     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01682             optimize->nend);
01683 #endif
01684
01685     // Calculating simulations to perform for each thread
01686     optimize->thread
01687         = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01688     for (i = 0; i <= nthreads; ++i)
01689     {
01690         optimize->thread[i] = optimize->nstart
01691             + i * (optimize->nend - optimize->nstart) / nthreads;
01692     #if DEBUG_OPTIMIZE
01693     fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01694             optimize->thread[i]);
01695     #endif
01696     }
01697     if (optimize->nsteps)
01698     {
01699         optimize->thread_direction = (unsigned int *)
01700             alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01701     }
01702
01703     // Opening result files
01704     optimize->file_result = g_fopen (optimize->result, "w");
01705     optimize->file_variables = g_fopen (optimize->variables, "w");
01706
01707     // Performing the algorithm
01708     switch (optimize->algorithm)
01709     {
01710     // Genetic algorithm
01711     case ALGORITHM_GENETIC:
01712         optimize_genetic ();
01713         break;
01714
01715     // Iterative algorithm
01716     default:
01717         optimize_iterate ();
01718     }
01719
01720     // Getting calculation time
01721     t = g_date_time_new_now (tz);
01722     optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01723     g_date_time_unref (t);
01724     g_date_time_unref (t0);
01725     g_time_zone_unref (tz);
01726     printf ("%s = %.6lg s\n",
01727             _("Calculation time"), optimize->calculation_time);
01728     fprintf (optimize->file_result, "%s = %.6lg s\n",
01729             _("Calculation time"), optimize->calculation_time);
01730
01731     // Closing result files
01732     fclose (optimize->file_variables);

```

```

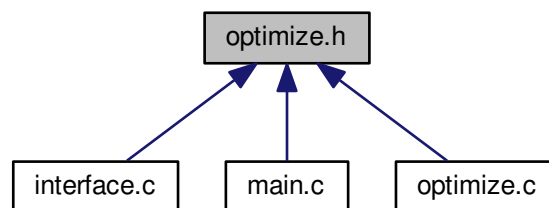
01728     fclose (optimize->file_result);
01729
01730     #if DEBUG_OPTIMIZE
01731     fprintf (stderr, "optimize_open: end\n");
01732     #endif
01733 }

```

4.19 optimize.h File Reference

Header file to define the optimization functions.

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



Data Structures

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

Functions

- void [optimize_input](#) (unsigned int simulation, char *[input](#), GMappedFile *[template](#))
Function to write the simulation input file.
- double [optimize_parse](#) (unsigned int simulation, unsigned int experiment)
Function to parse input files, simulating and calculating the \ objective function.
- double [optimize_norm_euclidian](#) (unsigned int simulation)
Function to calculate the Euclidian error norm.
- double [optimize_norm_maximum](#) (unsigned int simulation)
Function to calculate the maximum error norm.
- double [optimize_norm_p](#) (unsigned int simulation)
Function to calculate the P error norm.
- double [optimize_norm_taxicab](#) (unsigned int simulation)
Function to calculate the taxicab error norm.
- void [optimize_print](#) ()
Function to print the results.
- void [optimize_save_variables](#) (unsigned int simulation, double error)

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

 - void [optimize_best](#) (unsigned int simulation, double value)
- Function to save the best simulations.*

 - void [optimize_sequential](#) ()
- Function to optimize sequentially.*

 - void * [optimize_thread](#) ([ParallelData](#) *data)
- Function to optimize on a thread.*

 - void [optimize_merge](#) (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)
- Function to merge the 2 optimization results.*

 - void [optimize_synchronise](#) ()
- Function to synchronise the optimization results of MPI tasks.*

 - void [optimize_sweep](#) ()
- Function to optimize with the sweep algorithm.*

 - void [optimize_MonteCarlo](#) ()
- Function to optimize with the Monte-Carlo algorithm.*

 - void [optimize_best_direction](#) (unsigned int simulation, double value)
- Function to save the best simulation in a direction search method.*

 - void [optimize_direction_sequential](#) (unsigned int simulation)
- Function to estimate the direction search sequentially.*

 - void * [optimize_direction_thread](#) ([ParallelData](#) *data)
- Function to estimate the direction search on a thread.*

 - double [optimize_estimate_direction_random](#) (unsigned int variable, unsigned int estimate)
- Function to estimate a component of the direction search vector.*

 - double [optimize_estimate_direction_coordinates](#) (unsigned int variable, unsigned int estimate)
- Function to estimate a component of the direction search vector.*

 - void [optimize_step_direction](#) (unsigned int simulation)
- Function to do a step of the direction search method.*

 - void [optimize_direction](#) ()
- Function to optimize with a direction search method.*

 - double [optimize_genetic_objective](#) (**Entity** *entity)
- Function to calculate the objective function of an entity.*

 - void [optimize_genetic](#) ()
- Function to optimize with the genetic algorithm.*

 - void [optimize_save_old](#) ()
- Function to save the best results on iterative methods.*

 - void [optimize_merge_old](#) ()
- Function to merge the best results with the previous step best results on iterative methods.*

 - void [optimize_refine](#) ()
- Function to refine the search ranges of the variables in iterative algorithms.*

 - void [optimize_step](#) ()
- Function to do a step of the iterative algorithm.*

 - void [optimize_iterate](#) ()
- Function to iterate the algorithm.*

 - void [optimize_free](#) ()
- Function to free the memory used by the [Optimize](#) struct.*

 - void [optimize_open](#) ()
- Function to open and perform a optimization.*

Variables

- int [ntasks](#)
Number of tasks.
- unsigned int [nthreads](#)
Number of threads.
- unsigned int [nthreads_direction](#)
Number of threads for the direction search method.
- GMutex [mutex](#) [1]
Mutex struct.
- void(* [optimize_algorithm](#))()
Pointer to the function to perform a optimization algorithm step.
- double(* [optimize_estimate_direction](#))(unsigned int variable, unsigned int estimate)
Pointer to the function to estimate the direction.
- double(* [optimize_norm](#))(unsigned int simulation)
Pointer to the error norm function.
- [Optimize optimize](#) [1]
Optimization data.

4.19.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.19.2 Function Documentation

4.19.2.1 optimize_best()

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

Function to save the best simulations.

Parameters

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

Definition at line 468 of file `optimize.c`.

```

00469 {
00470     unsigned int i, j;
00471     double e;
00472     #if DEBUG_OPTIMIZE
00473         fprintf (stderr, "optimize_best: start\n");
00474         fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475                 optimize->nsaveds, optimize->nbest);
00476     #endif
00477     if (optimize->nsaveds < optimize->nbest
00478         || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480         if (optimize->nsaveds < optimize->nbest)
00481             ++optimize->nsaveds;
00482         optimize->error_best[optimize->nsaveds - 1] = value;
00483         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484         for (i = optimize->nsaveds; --i;)
00485         {
00486             if (optimize->error_best[i] < optimize->
00487                 error_best[i - 1])
00488             {
00489                 j = optimize->simulation_best[i];
00490                 e = optimize->error_best[i];
00491                 optimize->simulation_best[i] = optimize->
00492                     simulation_best[i - 1];
00493                 optimize->error_best[i] = optimize->
00494                     error_best[i - 1];
00495                 optimize->simulation_best[i - 1] = j;
00496                 optimize->error_best[i - 1] = e;
00497             }
00498             else
00499                 break;
00500         }
00501     }
00502     #if DEBUG_OPTIMIZE
00503         fprintf (stderr, "optimize_best: end\n");
00504     #endif
00505 }
```

4.19.2.2 optimize_best_direction()

```

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

Function to save the best simulation in a direction search method.

Parameters

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

Definition at line 795 of file `optimize.c`.

```

00796 {
00797     #if DEBUG_OPTIMIZE
00798         fprintf (stderr, "optimize_best_direction: start\n");
00799         fprintf (stderr,
00800                 "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801                 simulation, value, optimize->error_best[0]);
00802     #endif
00803     if (value < optimize->error_best[0])
00804     {
00805         optimize->error_best[0] = value;
00806         optimize->simulation_best[0] = simulation;
00807     }
00808     #if DEBUG_OPTIMIZE
```

```

00808     fprintf (stderr,
00809             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810             simulation, value);
00811 #endif
00812 }
00813 #if DEBUG_OPTIMIZE
00814 fprintf (stderr, "optimize_best_direction: end\n");
00815 #endif
00816 }

```

4.19.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

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

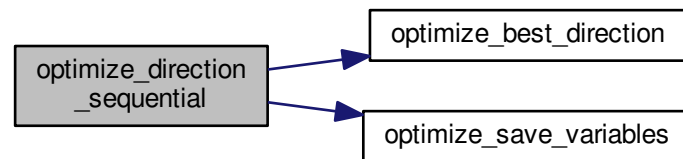
Definition at line 825 of file [optimize.c](#).

```

00826 {
00827     unsigned int i, j;
00828     double e;
00829 #if DEBUG_OPTIMIZE
00830     fprintf (stderr, "optimize_direction_sequential: start\n");
00831     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832             "nend_direction=%u\n",
00833             optimize->nstart_direction, optimize->
nend_direction);
00834 #endif
00835     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837         j = simulation + i;
00838         e = optimize_norm (j);
00839         optimize_best_direction (j, e);
00840         optimize_save_variables (j, e);
00841         if (e < optimize->threshold)
00842         {
00843             optimize->stop = 1;
00844             break;
00845         }
00846 #if DEBUG_OPTIMIZE
00847         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00848 #endif
00849     }
00850 #if DEBUG_OPTIMIZE
00851     fprintf (stderr, "optimize_direction_sequential: end\n");
00852 #endif
00853 }

```

Here is the call graph for this function:



4.19.2.4 optimize_direction_thread()

```
void* optimize_direction_thread (
    ParallelData * data )
```

Function to estimate the direction search on a thread.

Parameters

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

Returns

NULL

Definition at line 863 of file [optimize.c](#).

```

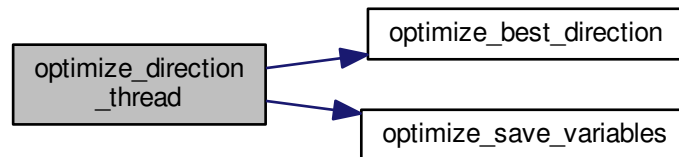
00864 {
00865     unsigned int i, thread;
00866     double e;
00867     #if DEBUG_OPTIMIZE
00868     fprintf (stderr, "optimize_direction_thread: start\n");
00869     #endif
00870     thread = data->thread;
00871     #if DEBUG_OPTIMIZE
00872     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873             thread,
00874             optimize->thread_direction[thread],
00875             optimize->thread_direction[thread + 1]);
00876     #endif
00877     for (i = optimize->thread_direction[thread];
00878          i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880         e = optimize_norm (i);
00881         g_mutex_lock (mutex);
00882         optimize_best_direction (i, e);
00883         optimize_save_variables (i, e);
00884         if (e < optimize->threshold)
00885             optimize->stop = 1;
00886         g_mutex_unlock (mutex);
00887         if (optimize->stop)
00888             break;
00889     #if DEBUG_OPTIMIZE
00890     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
  
```

```

00891 #endif
00892 }
00893 #if DEBUG_OPTIMIZE
00894     fprintf (stderr, "optimize_direction_thread: end\n");
00895 #endif
00896     g_thread_exit (NULL);
00897     return NULL;
00898 }

```

Here is the call graph for this function:



4.19.2.5 optimize_estimate_direction_coordinates()

```

double optimize_estimate_direction_coordinates (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 937 of file [optimize.c](#).

```

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

```


4.19.2.6 optimize_estimate_direction_random()

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

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 910 of file [optimize.c](#).

```
00912 {
00913     double x;
00914     #if DEBUG_OPTIMIZE
00915     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00916     #endif
00917     x = optimize->direction[variable]
00918         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00919         step[variable];
00919     #if DEBUG_OPTIMIZE
00920     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00921             variable, x);
00922     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00923     #endif
00924     return x;
00925 }
```

4.19.2.7 optimize_genetic_objective()

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

Function to calculate the objective function of an entity.

Parameters

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

Returns

objective function value.

Definition at line 1104 of file [optimize.c](#).

```
01105 {
01106     unsigned int j;
```

```

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

```

Here is the call graph for this function:



4.19.2.8 optimize_input()

```

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

```

Function to write the simulation input file.

Parameters

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

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

```

00105 {
00106     unsigned int i;
00107     char buffer[32], value[32], *buffer2, *buffer3, *content;
00108     FILE *file;

```

```

00109     gsize length;
00110     GRegex *regex;
00111
00112     #if DEBUG_OPTIMIZE
00113     fprintf (stderr, "optimize_input: start\n");
00114     #endif
00115
00116     // Checking the file
00117     if (!template)
00118         goto optimize_input_end;
00119
00120     // Opening template
00121     content = g_mapped_file_get_contents (template);
00122     length = g_mapped_file_get_length (template);
00123     #if DEBUG_OPTIMIZE
00124     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length,
00125             content);
00126     #endif
00127     file = g_fopen (input, "w");
00128
00129     // Parsing template
00130     for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132         #if DEBUG_OPTIMIZE
00133         fprintf (stderr, "optimize_input: variable=%u\n", i);
00134         #endif
00135         snprintf (buffer, 32, "@variable%u@", i + 1);
00136         regex = g_regex_new (buffer, 0, 0, NULL);
00137         if (i == 0)
00138         {
00139             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                               optimize->label[i], 0, NULL);
00141         #if DEBUG_OPTIMIZE
00142         fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143         #endif
00144         }
00145         else
00146         {
00147             length = strlen (buffer3);
00148             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                               optimize->label[i], 0, NULL);
00150             g_free (buffer3);
00151         }
00152         g_regex_unref (regex);
00153         length = strlen (buffer2);
00154         snprintf (buffer, 32, "@value%u@", i + 1);
00155         regex = g_regex_new (buffer, 0, 0, NULL);
00156         snprintf (value, 32, format[optimize->precision[i]],
00157                 optimize->value[simulation * optimize->
00158 nvariables + i]);
00159         #if DEBUG_OPTIMIZE
00160         fprintf (stderr, "optimize_input: value=%s\n", value);
00161         #endif
00162         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                           0, NULL);
00164         g_free (buffer2);
00165         g_regex_unref (regex);
00166     }
00167
00168     // Saving input file
00169     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170     g_free (buffer3);
00171     fclose (file);
00172
00173 optimize_input_end:
00174     #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176     #endif
00177     return;
00178 }

```

4.19.2.9 optimize_merge()

```

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

```

Function to merge the 2 optimization results.

Parameters

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

Definition at line 591 of file [optimize.c](#).

```

00593 {
00594     unsigned int i, j, k, s[optimize->nbest];
00595     double e[optimize->nbest];
00596     #if DEBUG_OPTIMIZE
00597     fprintf (stderr, "optimize_merge: start\n");
00598     #endif
00599     i = j = k = 0;
00600     do
00601     {
00602         if (i == optimize->nsaveds)
00603         {
00604             s[k] = simulation_best[j];
00605             e[k] = error_best[j];
00606             ++j;
00607             ++k;
00608             if (j == nsaveds)
00609                 break;
00610         }
00611         else if (j == nsaveds)
00612         {
00613             s[k] = optimize->simulation_best[i];
00614             e[k] = optimize->error_best[i];
00615             ++i;
00616             ++k;
00617             if (i == optimize->nsaveds)
00618                 break;
00619         }
00620         else if (optimize->error_best[i] > error_best[j])
00621         {
00622             s[k] = simulation_best[j];
00623             e[k] = error_best[j];
00624             ++j;
00625             ++k;
00626         }
00627         else
00628         {
00629             s[k] = optimize->simulation_best[i];
00630             e[k] = optimize->error_best[i];
00631             ++i;
00632             ++k;
00633         }
00634     }
00635     while (k < optimize->nbest);
00636     optimize->nsaveds = k;
00637     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638     memcpy (optimize->error_best, e, k * sizeof (double));
00639     #if DEBUG_OPTIMIZE
00640     fprintf (stderr, "optimize_merge: end\n");
00641     #endif
00642 }

```

4.19.2.10 optimize_norm_euclidian()

```

double optimize_norm_euclidian (
    unsigned int simulation )

```

Function to calculate the Euclidian error norm.

Parameters

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

Returns

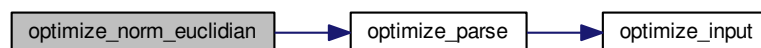
Euclidian error norm.

Definition at line 300 of file [optimize.c](#).

```

00301 {
00302     double e, ei;
00303     unsigned int i;
00304     #if DEBUG_OPTIMIZE
00305     fprintf (stderr, "optimize_norm_euclidian: start\n");
00306     #endif
00307     e = 0.;
00308     for (i = 0; i < optimize->nexperiments; ++i)
00309     {
00310         ei = optimize_parse (simulation, i);
00311         e += ei * ei;
00312     }
00313     e = sqrt (e);
00314     #if DEBUG_OPTIMIZE
00315     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316     fprintf (stderr, "optimize_norm_euclidian: end\n");
00317     #endif
00318     return e;
00319 }
```

Here is the call graph for this function:

**4.19.2.11 optimize_norm_maximum()**

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

Function to calculate the maximum error norm.

Parameters

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

Returns

Maximum error norm.

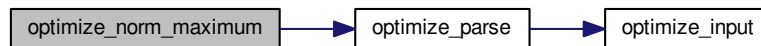
Definition at line 329 of file [optimize.c](#).

```

00330 {
00331     double e, ei;
00332     unsigned int i;
00333     #if DEBUG_OPTIMIZE
00334     fprintf (stderr, "optimize_norm_maximum: start\n");
00335     #endif
00336     e = 0.;
00337     for (i = 0; i < optimize->nexperiments; ++i)
00338     {
00339         ei = fabs (optimize_parse (simulation, i));
00340         e = fmax (e, ei);
00341     }
00342     #if DEBUG_OPTIMIZE
00343     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00344     fprintf (stderr, "optimize_norm_maximum: end\n");
00345     #endif
00346     return e;
00347 }

```

Here is the call graph for this function:



4.19.2.12 optimize_norm_p()

```

double optimize_norm_p (
    unsigned int simulation )

```

Function to calculate the P error norm.

Parameters

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

Returns

P error norm.

Definition at line 357 of file [optimize.c](#).

```

00358 {
00359     double e, ei;
00360     unsigned int i;
00361     #if DEBUG_OPTIMIZE
00362     fprintf (stderr, "optimize_norm_p: start\n");
00363     #endif
00364     e = 0.;
00365     for (i = 0; i < optimize->nexperiments; ++i)
00366     {
00367         ei = fabs (optimize_parse (simulation, i));

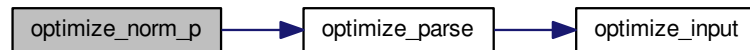
```

```

00368     e += pow (ei, optimize->p);
00369 }
00370 e = pow (e, 1. / optimize->p);
00371 #if DEBUG_OPTIMIZE
00372 fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00373 fprintf (stderr, "optimize_norm_p: end\n");
00374 #endif
00375 return e;
00376 }

```

Here is the call graph for this function:



4.19.2.13 optimize_norm_taxicab()

```

double optimize_norm_taxicab (
    unsigned int simulation )

```

Function to calculate the taxicab error norm.

Parameters

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

Returns

Taxicab error norm.

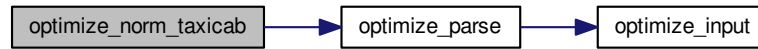
Definition at line 386 of file [optimize.c](#).

```

00387 {
00388     double e;
00389     unsigned int i;
00390     #if DEBUG_OPTIMIZE
00391     fprintf (stderr, "optimize_norm_taxicab: start\n");
00392     #endif
00393     e = 0.;
00394     for (i = 0; i < optimize->nexperiments; ++i)
00395         e += fabs (optimize_parse (simulation, i));
00396     #if DEBUG_OPTIMIZE
00397     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398     fprintf (stderr, "optimize_norm_taxicab: end\n");
00399     #endif
00400     return e;
00401 }

```


Here is the call graph for this function:



4.19.2.14 optimize_parse()

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

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

Parameters

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

Returns

Objective function value.

Definition at line 191 of file [optimize.c](#).

```

00192 {
00193     unsigned int i;
00194     double e;
00195     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196         *buffer3, *buffer4;
00197     FILE *file_result;
00198
00199     #if DEBUG_OPTIMIZE
00200         fprintf (stderr, "optimize_parse: start\n");
00201         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202             simulation, experiment);
00203     #endif
00204
00205     // Opening input files
00206     for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209             experiment);
00210         #if DEBUG_OPTIMIZE
00211             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212         #endif
00213         optimize_input (simulation, &input[i][0],
00214             optimize->file[i][experiment]);
00215     }
00216     for (; i < MAX_NINPUTS; ++i)
00217         strcpy (&input[i][0], "");
00218     #if DEBUG_OPTIMIZE
00219         fprintf (stderr, "optimize_parse: parsing end\n");
00220     #endif
00221
00222     // Performing the simulation

```

```

00223     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224     buffer2 = g_path_get_dirname (optimize->simulator);
00225     buffer3 = g_path_get_basename (optimize->simulator);
00226     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230     g_free (buffer4);
00231     g_free (buffer3);
00232     g_free (buffer2);
00233     #if DEBUG_OPTIMIZE
00234     fprintf (stderr, "optimize_parse: %s\n", buffer);
00235     #endif
00236     system (buffer);
00237
00238     // Checking the objective value function
00239     if (optimize->evaluator)
00240     {
00241         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242         buffer2 = g_path_get_dirname (optimize->evaluator);
00243         buffer3 = g_path_get_basename (optimize->evaluator);
00244         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245         snprintf (buffer, 512, "\"%s\" %s %s %s",
00246                 buffer4, output, optimize->experiment[experiment], result);
00247         g_free (buffer4);
00248         g_free (buffer3);
00249         g_free (buffer2);
00250         #if DEBUG_OPTIMIZE
00251         fprintf (stderr, "optimize_parse: %s\n", buffer);
00252         #endif
00253         system (buffer);
00254         file_result = g_fopen (result, "r");
00255         e = atof (fgets (buffer, 512, file_result));
00256         fclose (file_result);
00257     }
00258     else
00259     {
00260         strcpy (result, "");
00261         file_result = g_fopen (output, "r");
00262         e = atof (fgets (buffer, 512, file_result));
00263         fclose (file_result);
00264     }
00265
00266     // Removing files
00267     #if !DEBUG_OPTIMIZE
00268     for (i = 0; i < optimize->ninputs; ++i)
00269     {
00270         if (optimize->file[i][0])
00271         {
00272             snprintf (buffer, 512, RM " %s", &input[i][0]);
00273             system (buffer);
00274         }
00275     }
00276     snprintf (buffer, 512, RM " %s %s", output, result);
00277     system (buffer);
00278     #endif
00279
00280     // Processing pending events
00281     if (show_pending)
00282         show_pending ();
00283
00284     #if DEBUG_OPTIMIZE
00285     fprintf (stderr, "optimize_parse: end\n");
00286     #endif
00287
00288     // Returning the objective function
00289     return e * optimize->weight[experiment];
00290 }

```

Here is the call graph for this function:



4.19.2.15 optimize_save_variables()

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

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

Parameters

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

Definition at line 439 of file [optimize.c](#).

```
00440 {
00441     unsigned int i;
00442     char buffer[64];
00443     #if DEBUG_OPTIMIZE
00444     fprintf (stderr, "optimize_save_variables: start\n");
00445     #endif
00446     for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449         fprintf (optimize->file_variables, buffer,
00450             optimize->value[simulation * optimize->
00451                 nvariables + i]);
00452         fprintf (optimize->file_variables, "%.14le\n", error);
00453         fflush (optimize->file_variables);
00454     #if DEBUG_OPTIMIZE
00455     fprintf (stderr, "optimize_save_variables: end\n");
00456     #endif
00457 }
```

4.19.2.16 optimize_step_direction()

```
void optimize_step_direction (
    unsigned int simulation )
```

Function to do a step of the direction search method.

Parameters

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

Definition at line 968 of file [optimize.c](#).

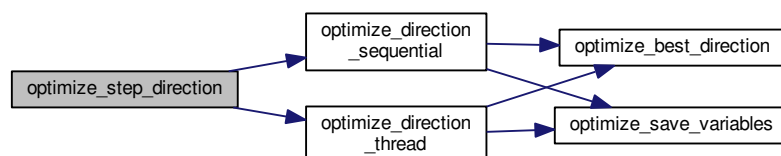
```
00969 {
00970     GThread *thread[nthreads_direction];
00971     ParallelData data[nthreads_direction];
00972     unsigned int i, j, k, b;
00973     #if DEBUG_OPTIMIZE
```

```

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

```

Here is the call graph for this function:



4.19.2.17 optimize_thread()

```
void* optimize_thread (
    ParallelData * data )
```

Function to optimize on a thread.

Parameters

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

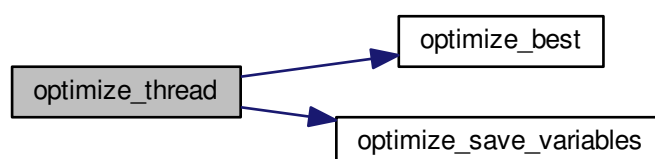
Returns

NULL

Definition at line 545 of file [optimize.c](#).

```
00546 {
00547     unsigned int i, thread;
00548     double e;
00549     #if DEBUG_OPTIMIZE
00550     fprintf (stderr, "optimize_thread: start\n");
00551     #endif
00552     thread = data->thread;
00553     #if DEBUG_OPTIMIZE
00554     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00555             optimize->thread[thread], optimize->thread[thread + 1]);
00556     #endif
00557     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00558     {
00559         e = optimize_norm (i);
00560         g_mutex_lock (mutex);
00561         optimize_best (i, e);
00562         optimize_save_variables (i, e);
00563         if (e < optimize->threshold)
00564             optimize->stop = 1;
00565         g_mutex_unlock (mutex);
00566         if (optimize->stop)
00567             break;
00568     #if DEBUG_OPTIMIZE
00569     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570     #endif
00571     }
00572     #if DEBUG_OPTIMIZE
00573     fprintf (stderr, "optimize_thread: end\n");
00574     #endif
00575     g_thread_exit (NULL);
00576     return NULL;
00577 }
```

Here is the call graph for this function:



4.20 optimize.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, 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 *direction;
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_direction;
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_direction;
00084     unsigned int nend_direction;

```

```

00109 unsigned int niterations;
00110 unsigned int nbest;
00111 unsigned int nsaveds;
00112 unsigned int stop;
00113 #if HAVE_MPI
00114 int mpi_rank;
00115 #endif
00116 } Optimize;
00117
00122 typedef struct
00123 {
00124 unsigned int thread;
00125 } ParallelData;
00126
00127 // Global variables
00128 extern int ntasks;
00129 extern unsigned int nthreads;
00130 extern unsigned int nthreads_direction;
00131 extern GMutex mutex[1];
00132 extern void (*optimize_algorithm) ();
00133 extern double (*optimize_estimate_direction) (unsigned int variable,
00134 unsigned int estimate);
00135 extern double (*optimize_norm) (unsigned int simulation);
00136 extern Optimize optimize[1];
00137
00138 // Public functions
00139 void optimize_input (unsigned int simulation, char *input,
00140 GMappedFile * template);
00141 double optimize_parse (unsigned int simulation, unsigned int experiment);
00142 double optimize_norm_euclidian (unsigned int simulation);
00143 double optimize_norm_maximum (unsigned int simulation);
00144 double optimize_norm_p (unsigned int simulation);
00145 double optimize_norm_taxicab (unsigned int simulation);
00146 void optimize_print ();
00147 void optimize_save_variables (unsigned int simulation, double error);
00148 void optimize_best (unsigned int simulation, double value);
00149 void optimize_sequential ();
00150 void *optimize_thread (ParallelData * data);
00151 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00152 double *error_best);
00153 #if HAVE_MPI
00154 void optimize_synchronise ();
00155 #endif
00156 void optimize_sweep ();
00157 void optimize_MonteCarlo ();
00158 void optimize_best_direction (unsigned int simulation, double value);
00159 void optimize_direction_sequential (unsigned int simulation);
00160 void *optimize_direction_thread (ParallelData * data);
00161 double optimize_estimate_direction_random (unsigned int variable,
00162 unsigned int estimate);
00163 double optimize_estimate_direction_coordinates (unsigned int
variable,
00164 unsigned int estimate);
00165 void optimize_step_direction (unsigned int simulation);
00166 void optimize_direction ();
00167 double optimize_genetic_objective (Entity * entity);
00168 void optimize_genetic ();
00169 void optimize_save_old ();
00170 void optimize_merge_old ();
00171 void optimize_refine ();
00172 void optimize_step ();
00173 void optimize_iterate ();
00174 void optimize_free ();
00175 void optimize_open ();
00176
00177 #endif

```

4.21 utils.c File Reference

Source file to define some useful functions.

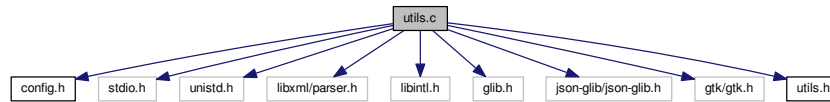
```

#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)
Function to show a dialog with a message.
- void [show_error](#) (char *msg)
Function to show a dialog with an error message.
- int [xml_node_get_int](#) (xmlNode *node, const xmlChar *prop, int *error_code)
Function to get an integer number of a XML node property.
- 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.
- 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.
- 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.
- 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.
- void [xml_node_set_int](#) (xmlNode *node, const xmlChar *prop, int value)
Function to set an integer number in a XML node property.
- 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.
- void [xml_node_set_float](#) (xmlNode *node, const xmlChar *prop, double value)
Function to set a floating point number in a XML node property.
- int [json_object_get_int](#) (JsonObject *object, const char *prop, int *error_code)
Function to get an integer number of a JSON object property.
- 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.
- 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.
- 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.
- 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.
- void [json_object_set_int](#) (JsonObject *object, const char *prop, int value)
Function to set an integer number in a JSON object property.
- 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.*
- void [json_object_set_float](#) (JsonObject *object, const char *prop, double value)
Function to set a floating point number in a JSON object property.
- int [cores_number](#) ()
Function to obtain the cores number.
- void [process_pending](#) ()
Function to process events on long computation.
- unsigned int [gtk_array_get_active](#) (GtkRadioButton *array[], unsigned int n)
Function to get the active GtkRadioButton.

Variables

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

4.21.1 Detailed Description

Source file to define some useful functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.21.2 Function Documentation

4.21.2.1 [cores_number\(\)](#)

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line [532](#) of file [utils.c](#).

```
00533 {
00534 #ifdef G_OS_WIN32
00535     SYSTEM_INFO sysinfo;
00536     GetSystemInfo (&sysinfo);
00537     return sysinfo.dwNumberOfProcessors;
00538 #else
00539     return (int) sysconf (_SC_NPROCESSORS_ONLN);
00540 #endif
00541 }
```

4.21.2.2 gtk_array_get_active()

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

Function to get the active GtkWidget.

Parameters

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

Returns

Active GtkWidget.

Definition at line 567 of file [utils.c](#).

```
00568 {  
00569     unsigned int i;  
00570     for (i = 0; i < n; ++i)  
00571         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))  
00572             break;  
00573     return i;  
00574 }
```

4.21.2.3 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.

Parameters

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

Returns

Floating point number value.

Definition at line 421 of file [utils.c](#).

```
00422 {
```

```
00423  const char *buffer;
00424  double x = 0.;
00425  buffer = json_object_get_string_member (object, prop);
00426  if (!buffer)
00427      *error_code = 1;
00428  else
00429      {
00430          if (sscanf (buffer, "%lf", &x) != 1)
00431              *error_code = 2;
00432          else
00433              *error_code = 0;
00434      }
00435  return x;
00436 }
```

4.21.2.4 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.

Parameters

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

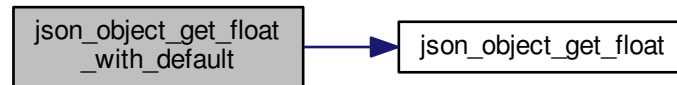
Returns

Floating point number value.

Definition at line [454](#) of file [utils.c](#).

```
00456 {
00457     double x;
00458     if (json_object_get_member (object, prop))
00459         x = json_object_get_float (object, prop, error_code);
00460     else
00461     {
00462         x = default_value;
00463         *error_code = 0;
00464     }
00465     return x;
00466 }
```

Here is the call graph for this function:



4.21.2.5 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.

Parameters

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

Returns

Integer number value.

Definition at line 330 of file [utils.c](#).

```
00331 {  
00332     const char *buffer;  
00333     int i = 0;  
00334     buffer = json_object_get_string_member (object, prop);  
00335     if (!buffer)  
00336         *error_code = 1;  
00337     else  
00338     {  
00339         if (sscanf (buffer, "%d", &i) != 1)  
00340             *error_code = 2;  
00341         else  
00342             *error_code = 0;  
00343     }  
00344     return i;  
00345 }
```

4.21.2.6 json_object_get_uint()

```
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.

Parameters

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

Returns

Unsigned integer number value.

Definition at line 360 of file [utils.c](#).

```
00361 {
00362     const char *buffer;
00363     unsigned int i = 0;
00364     buffer = json_object_get_string_member (object, prop);
00365     if (!buffer)
00366         *error_code = 1;
00367     else
00368     {
00369         if (sscanf (buffer, "%u", &i) != 1)
00370             *error_code = 2;
00371         else
00372             *error_code = 0;
00373     }
00374     return i;
00375 }
```

4.21.2.7 json_object_get_uint_with_default()

```
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.

Parameters

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

Returns

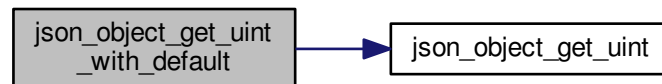
Unsigned integer number value.

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

```

00396 {
00397     unsigned int i;
00398     if (json_object_get_member (object, prop))
00399         i = json_object_get_uint (object, prop, error_code);
00400     else
00401     {
00402         i = default_value;
00403         *error_code = 0;
00404     }
00405     return i;
00406 }
```

Here is the call graph for this function:

**4.21.2.8 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 519 of file [utils.c](#).

```

00520 {
00521     char buffer[64];
00522     snprintf (buffer, 64, "%.14lg", value);
00523     json_object_set_string_member (object, prop, buffer);
00524 }
```

4.21.2.9 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 [480](#) of file [utils.c](#).

```
00481 {
00482     char buffer[64];
00483     snprintf (buffer, 64, "%d", value);
00484     json_object_set_string_member (object, prop, buffer);
00485 }
```

4.21.2.10 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 [499](#) of file [utils.c](#).

```
00501 {
00502     char buffer[64];
00503     snprintf (buffer, 64, "%u", value);
00504     json_object_set_string_member (object, prop, buffer);
00505 }
```

4.21.2.11 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 103 of file [utils.c](#).

```
00104 {
00105     show\_message (_("ERROR!"), msg, ERROR\_TYPE);
00106 }
```

Here is the call graph for this function:



4.21.2.12 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 73 of file [utils.c](#).

```
00074 {
00075     #if HAVE\_GTK
```



```

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

```

4.21.2.13 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.

Parameters

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

Returns

Floating point number value.

Definition at line 213 of file [utils.c](#).

```

00214 {
00215     double x = 0.;
00216     xmlChar *buffer;
00217     buffer = xmlGetProp (node, prop);
00218     if (!buffer)
00219         *error_code = 1;
00220     else
00221     {
00222         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223             *error_code = 2;
00224         else
00225             *error_code = 0;
00226         xmlFree (buffer);
00227     }
00228     return x;
00229 }

```

4.21.2.14 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.

Parameters

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

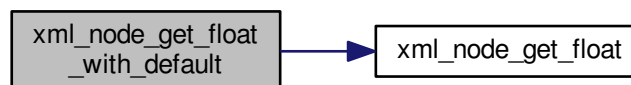
Returns

Floating point number value.

Definition at line 247 of file [utils.c](#).

```
00249 {
00250     double x;
00251     if (xmlHasProp (node, prop))
00252         x = xml_node_get_float (node, prop, error_code);
00253     else
00254     {
00255         x = default_value;
00256         *error_code = 0;
00257     }
00258     return x;
00259 }
```

Here is the call graph for this function:



4.21.2.15 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.

Parameters

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

Returns

Integer number value.

Definition at line 121 of file [utils.c](#).

```
00122 {
00123     int i = 0;
00124     xmlChar *buffer;
00125     buffer = xmlGetProp (node, prop);
00126     if (!buffer)
00127         *error_code = 1;
00128     else
00129     {
00130         if (sscanf ((char *) buffer, "%d", &i) != 1)
00131             *error_code = 2;
00132         else
00133             *error_code = 0;
00134         xmlFree (buffer);
00135     }
00136     return i;
00137 }
```

4.21.2.16 xml_node_get_uint()

```
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.

Parameters

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

Returns

Unsigned integer number value.

Definition at line 152 of file [utils.c](#).

```
00153 {
00154     unsigned int i = 0;
00155     xmlChar *buffer;
00156     buffer = xmlGetProp (node, prop);
```

```

00157  if (!buffer)
00158      *error_code = 1;
00159  else
00160      {
00161          if (sscanf ((char *) buffer, "%u", &i) != 1)
00162              *error_code = 2;
00163          else
00164              *error_code = 0;
00165          xmlFree (buffer);
00166      }
00167  return i;
00168 }

```

4.21.2.17 xml_node_get_uint_with_default()

```

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.

Parameters

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

Returns

Unsigned integer number value.

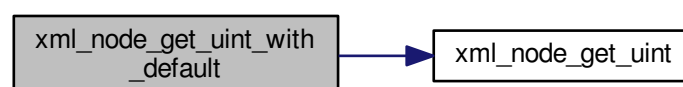
Definition at line 186 of file [utils.c](#).

```

00188 {
00189     unsigned int i;
00190     if (xmlHasProp (node, prop))
00191         i = xml_node_get_uint (node, prop, error_code);
00192     else
00193     {
00194         i = default_value;
00195         *error_code = 0;
00196     }
00197     return i;
00198 }

```

Here is the call graph for this function:



4.21.2.18 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 310 of file [utils.c](#).

```
00311 {
00312     xmlChar buffer[64];
00313     snprintf ((char *) buffer, 64, "%.14lg", value);
00314     xmlSetProp (node, prop, buffer);
00315 }
```

4.21.2.19 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 272 of file [utils.c](#).

```
00273 {
00274     xmlChar buffer[64];
00275     snprintf ((char *) buffer, 64, "%d", value);
00276     xmlSetProp (node, prop, buffer);
00277 }
```

4.21.2.20 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 291 of file [utils.c](#).

```
00292 {
00293     xmlChar buffer[64];
00294     snprintf ((char *) buffer, 64, "%u", value);
00295     xmlSetProp (node, prop, buffer);
00296 }
```

4.22 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 Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, 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>
```

```

00051 #endif
00052 #include "utils.h"
00053
00054 #if HAVE_GTK
00055 GtkWidget *main_window;
00056 #endif
00057
00058 char *error_message;
00059 void (*show_pending) () = NULL;
00061
00072 void
00073 show_message (char *title, char *msg, int type)
00074 {
00075     #if HAVE_GTK
00076         GtkMessageDialog *dlg;
00077
00078         // Creating the dialog
00079         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00080             (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00081
00082         // Setting the dialog title
00083         gtk_window_set_title (GTK_WINDOW (dlg), title);
00084
00085         // Showing the dialog and waiting response
00086         gtk_dialog_run (GTK_DIALOG (dlg));
00087
00088         // Closing and freeing memory
00089         gtk_widget_destroy (GTK_WIDGET (dlg));
00090
00091     #else
00092         printf ("%s: %s\n", title, msg);
00093     #endif
00094 }
00095
00102 void
00103 show_error (char *msg)
00104 {
00105     show_message (_("ERROR!"), msg, ERROR_TYPE);
00106 }
00107
00120 int
00121 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00122 {
00123     int i = 0;
00124     xmlChar *buffer;
00125     buffer = xmlGetProp (node, prop);
00126     if (!buffer)
00127         *error_code = 1;
00128     else
00129     {
00130         if (sscanf ((char *) buffer, "%d", &i) != 1)
00131             *error_code = 2;
00132         else
00133             *error_code = 0;
00134         xmlFree (buffer);
00135     }
00136     return i;
00137 }
00138
00151 unsigned int
00152 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00153 {
00154     unsigned int i = 0;
00155     xmlChar *buffer;
00156     buffer = xmlGetProp (node, prop);
00157     if (!buffer)
00158         *error_code = 1;
00159     else
00160     {
00161         if (sscanf ((char *) buffer, "%u", &i) != 1)
00162             *error_code = 2;
00163         else
00164             *error_code = 0;
00165         xmlFree (buffer);
00166     }
00167     return i;
00168 }
00169
00185 unsigned int
00186 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00187                                unsigned int default_value, int *error_code)
00188 {
00189     unsigned int i;
00190     if (xmlHasProp (node, prop))
00191         i = xml_node_get_uint (node, prop, error_code);
00192     else
00193     {

```

```

00194         i = default_value;
00195         *error_code = 0;
00196     }
00197     return i;
00198 }
00199
00212 double
00213 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00214 {
00215     double x = 0.;
00216     xmlChar *buffer;
00217     buffer = xmlGetProp (node, prop);
00218     if (!buffer)
00219         *error_code = 1;
00220     else
00221     {
00222         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223             *error_code = 2;
00224         else
00225             *error_code = 0;
00226         xmlFree (buffer);
00227     }
00228     return x;
00229 }
00230
00246 double
00247 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00248                                 double default_value, int *error_code)
00249 {
00250     double x;
00251     if (xmlHasProp (node, prop))
00252         x = xml_node_get_float (node, prop, error_code);
00253     else
00254     {
00255         x = default_value;
00256         *error_code = 0;
00257     }
00258     return x;
00259 }
00260
00271 void
00272 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00273 {
00274     xmlChar buffer[64];
00275     snprintf ((char *) buffer, 64, "%d", value);
00276     xmlSetProp (node, prop, buffer);
00277 }
00278
00290 void
00291 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00292 {
00293     xmlChar buffer[64];
00294     snprintf ((char *) buffer, 64, "%u", value);
00295     xmlSetProp (node, prop, buffer);
00296 }
00297
00309 void
00310 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00311 {
00312     xmlChar buffer[64];
00313     snprintf ((char *) buffer, 64, "%.14lg", value);
00314     xmlSetProp (node, prop, buffer);
00315 }
00316
00329 int
00330 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00331 {
00332     const char *buffer;
00333     int i = 0;
00334     buffer = json_object_get_string_member (object, prop);
00335     if (!buffer)
00336         *error_code = 1;
00337     else
00338     {
00339         if (sscanf (buffer, "%d", &i) != 1)
00340             *error_code = 2;
00341         else
00342             *error_code = 0;
00343     }
00344     return i;
00345 }
00346
00359 unsigned int
00360 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00361 {
00362     const char *buffer;
00363     unsigned int i = 0;

```



```

00364     buffer = json_object_get_string_member (object, prop);
00365     if (!buffer)
00366         *error_code = 1;
00367     else
00368     {
00369         if (sscanf (buffer, "%u", &i) != 1)
00370             *error_code = 2;
00371         else
00372             *error_code = 0;
00373     }
00374     return i;
00375 }
00376
00392 unsigned int
00393 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00394                                   unsigned int default_value,
00395                                   int *error_code)
00396 {
00397     unsigned int i;
00398     if (json_object_get_member (object, prop))
00399         i = json_object_get_uint (object, prop, error_code);
00400     else
00401     {
00402         i = default_value;
00403         *error_code = 0;
00404     }
00405     return i;
00406 }
00407
00420 double
00421 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00422 {
00423     const char *buffer;
00424     double x = 0.;
00425     buffer = json_object_get_string_member (object, prop);
00426     if (!buffer)
00427         *error_code = 1;
00428     else
00429     {
00430         if (sscanf (buffer, "%lf", &x) != 1)
00431             *error_code = 2;
00432         else
00433             *error_code = 0;
00434     }
00435     return x;
00436 }
00437
00453 double
00454 json_object_get_float_with_default (JsonObject * object, const char *prop,
00455                                     double default_value, int *error_code)
00456 {
00457     double x;
00458     if (json_object_get_member (object, prop))
00459         x = json_object_get_float (object, prop, error_code);
00460     else
00461     {
00462         x = default_value;
00463         *error_code = 0;
00464     }
00465     return x;
00466 }
00467
00479 void
00480 json_object_set_int (JsonObject * object, const char *prop, int value)
00481 {
00482     char buffer[64];
00483     snprintf (buffer, 64, "%d", value);
00484     json_object_set_string_member (object, prop, buffer);
00485 }
00486
00498 void
00499 json_object_set_uint (JsonObject * object, const char *prop,
00500                      unsigned int value)
00501 {
00502     char buffer[64];
00503     snprintf (buffer, 64, "%u", value);
00504     json_object_set_string_member (object, prop, buffer);
00505 }
00506
00518 void
00519 json_object_set_float (JsonObject * object, const char *prop, double value)
00520 {
00521     char buffer[64];
00522     snprintf (buffer, 64, "%.14lg", value);
00523     json_object_set_string_member (object, prop, buffer);
00524 }

```

```

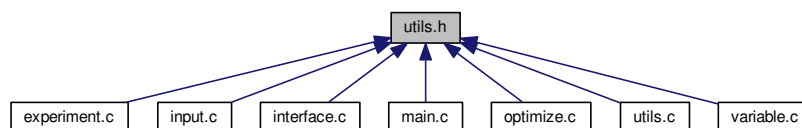
00525
00531 int
00532 cores_number ()
00533 {
00534     #ifdef G_OS_WIN32
00535         SYSTEM_INFO sysinfo;
00536         GetSystemInfo (&sysinfo);
00537         return sysinfo.dwNumberOfProcessors;
00538     #else
00539         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00540     #endif
00541 }
00542
00543 #if HAVE_GTK
00544
00549 void
00550 process_pending ()
00551 {
00552     while (gtk_events_pending ())
00553         gtk_main_iteration ();
00554 }
00555
00566 unsigned int
00567 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00568 {
00569     unsigned int i;
00570     for (i = 0; i < n; ++i)
00571         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572             break;
00573     return i;
00574 }
00575
00576 #endif

```

4.23 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)
Function to show a dialog with a message.
- void `show_error` (char *msg)
Function to show a dialog with an error message.

- int [xml_node_get_int](#) (xmlNode *node, const xmlChar *prop, int *error_code)
Function to get an integer number of a XML node property.
- 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.
- 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.
- 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.
- 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.
- void [xml_node_set_int](#) (xmlNode *node, const xmlChar *prop, int value)
Function to set an integer number in a XML node property.
- 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.
- void [xml_node_set_float](#) (xmlNode *node, const xmlChar *prop, double value)
Function to set a floating point number in a XML node property.
- int [json_object_get_int](#) (JsonObject *object, const char *prop, int *error_code)
Function to get an integer number of a JSON object property.
- 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.
- 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.
- 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.
- 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.
- void [json_object_set_int](#) (JsonObject *object, const char *prop, int value)
Function to set an integer number in a JSON object property.
- 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.
- void [json_object_set_float](#) (JsonObject *object, const char *prop, double value)
Function to set a floating point number in a JSON object property.
- int [cores_number](#) ()
Function to obtain the cores number.
- void [process_pending](#) ()
Function to process events on long computation.
- unsigned int [gtk_array_get_active](#) (GtkRadioButton *array[], unsigned int n)
Function to get the active GtkRadioButton.

Variables

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

4.23.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.23.2 Function Documentation

4.23.2.1 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line [532](#) of file [utils.c](#).

```
00533 {  
00534     #ifdef G_OS_WIN32  
00535         SYSTEM_INFO sysinfo;  
00536         GetSystemInfo (&sysinfo);  
00537         return sysinfo.dwNumberOfProcessors;  
00538     #else  
00539         return (int) sysconf (_SC_NPROCESSORS_ONLN);  
00540     #endif  
00541 }
```

4.23.2.2 gtk_array_get_active()

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

Function to get the active GtkRadioButton.

Parameters

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

Returns

Active GtkRadioButton.

Definition at line 567 of file [utils.c](#).

```
00568 {
00569     unsigned int i;
00570     for (i = 0; i < n; ++i)
00571         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572             break;
00573     return i;
00574 }
```

4.23.2.3 json_object_get_float()

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 421 of file [utils.c](#).

```
00422 {
00423     const char *buffer;
00424     double x = 0.;
00425     buffer = json_object_get_string_member (object, prop);
00426     if (!buffer)
00427         *error_code = 1;
00428     else
00429     {
00430         if (sscanf (buffer, "%lf", &x) != 1)
00431             *error_code = 2;
00432         else
00433             *error_code = 0;
00434     }
00435     return x;
00436 }
```

4.23.2.4 json_object_get_float_with_default()

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

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

Parameters

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

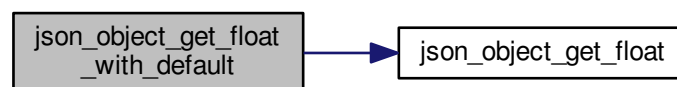
Returns

Floating point number value.

Definition at line [454](#) of file [utils.c](#).

```
00456 {
00457     double x;
00458     if (json_object_get_member (object, prop))
00459         x = json_object_get_float (object, prop, error_code);
00460     else
00461     {
00462         x = default_value;
00463         *error_code = 0;
00464     }
00465     return x;
00466 }
```

Here is the call graph for this function:



4.23.2.5 json_object_get_int()

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

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

Parameters

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

Returns

Integer number value.

Definition at line 330 of file [utils.c](#).

```
00331 {
00332     const char *buffer;
00333     int i = 0;
00334     buffer = json_object_get_string_member (object, prop);
00335     if (!buffer)
00336         *error_code = 1;
00337     else
00338     {
00339         if (sscanf (buffer, "%d", &i) != 1)
00340             *error_code = 2;
00341         else
00342             *error_code = 0;
00343     }
00344     return i;
00345 }
```

4.23.2.6 json_object_get_uint()

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 360 of file [utils.c](#).

```
00361 {
00362     const char *buffer;
00363     unsigned int i = 0;
00364     buffer = json_object_get_string_member (object, prop);
```

```

00365  if (!buffer)
00366      *error_code = 1;
00367  else
00368      {
00369          if (sscanf (buffer, "%u", &i) != 1)
00370              *error_code = 2;
00371          else
00372              *error_code = 0;
00373      }
00374  return i;
00375  }

```

4.23.2.7 json_object_get_uint_with_default()

```

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

```

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

Parameters

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

Returns

Unsigned integer number value.

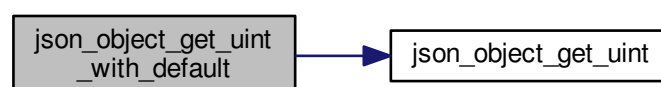
Definition at line 393 of file [utils.c](#).

```

00396  {
00397      unsigned int i;
00398      if (json_object_get_member (object, prop))
00399          i = json_object_get_uint (object, prop, error_code);
00400      else
00401          {
00402              i = default_value;
00403              *error_code = 0;
00404          }
00405      return i;
00406  }

```

Here is the call graph for this function:



4.23.2.8 json_object_set_float()

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

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

Parameters

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

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

```
00520 {
00521     char buffer[64];
00522     snprintf (buffer, 64, "%.14lg", value);
00523     json_object_set_string_member (object, prop, buffer);
00524 }
```

4.23.2.9 json_object_set_int()

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

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

Parameters

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

Definition at line 480 of file [utils.c](#).

```
00481 {
00482     char buffer[64];
00483     snprintf (buffer, 64, "%d", value);
00484     json_object_set_string_member (object, prop, buffer);
00485 }
```

4.23.2.10 json_object_set_uint()

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

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

Parameters

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

Definition at line [499](#) of file [utils.c](#).

```
00501 {
00502     char buffer[64];
00503     snprintf (buffer, 64, "%u", value);
00504     json_object_set_string_member (object, prop, buffer);
00505 }
```

4.23.2.11 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 [103](#) of file [utils.c](#).

```
00104 {
00105     show_message (_("ERROR!"), msg, ERROR_TYPE);
00106 }
```

Here is the call graph for this function:



4.23.2.12 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 73 of file [utils.c](#).

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

4.23.2.13 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.

Parameters

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

Returns

Floating point number value.

Definition at line 213 of file [utils.c](#).

```

00214 {
00215     double x = 0.;
00216     xmlChar *buffer;
00217     buffer = xmlGetProp (node, prop);
00218     if (!buffer)
00219         *error_code = 1;
00220     else
00221     {
00222         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223             *error_code = 2;
00224         else
00225             *error_code = 0;
00226         xmlFree (buffer);
00227     }
00228     return x;
00229 }
```

4.23.2.14 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.

Parameters

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

Returns

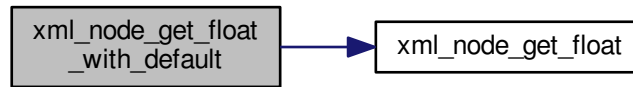
Floating point number value.

Definition at line 247 of file [utils.c](#).

```

00249 {
00250     double x;
00251     if (xmlHasProp (node, prop))
00252         x = xml_node_get_float (node, prop, error_code);
00253     else
00254     {
00255         x = default_value;
00256         *error_code = 0;
00257     }
00258     return x;
00259 }
```

Here is the call graph for this function:



4.23.2.15 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.

Parameters

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

Returns

Integer number value.

Definition at line 121 of file [utils.c](#).

```
00122 {  
00123     int i = 0;  
00124     xmlChar *buffer;  
00125     buffer = xmlGetProp (node, prop);  
00126     if (!buffer)  
00127         *error_code = 1;  
00128     else  
00129     {  
00130         if (sscanf ((char *) buffer, "%d", &i) != 1)  
00131             *error_code = 2;  
00132         else  
00133             *error_code = 0;  
00134         xmlFree (buffer);  
00135     }  
00136     return i;  
00137 }
```

4.23.2.16 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.

Parameters

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

Returns

Unsigned integer number value.

Definition at line 152 of file [utils.c](#).

```
00153 {
00154     unsigned int i = 0;
00155     xmlChar *buffer;
00156     buffer = xmlGetProp (node, prop);
00157     if (!buffer)
00158         *error_code = 1;
00159     else
00160     {
00161         if (sscanf ((char *) buffer, "%u", &i) != 1)
00162             *error_code = 2;
00163         else
00164             *error_code = 0;
00165         xmlFree (buffer);
00166     }
00167     return i;
00168 }
```

4.23.2.17 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.

Parameters

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

Returns

Unsigned integer number value.

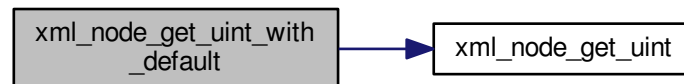
Definition at line 186 of file [utils.c](#).

```

00188 {
00189     unsigned int i;
00190     if (xmlHasProp (node, prop))
00191         i = xml_node_get_uint (node, prop, error_code);
00192     else
00193     {
00194         i = default_value;
00195         *error_code = 0;
00196     }
00197     return i;
00198 }

```

Here is the call graph for this function:

**4.23.2.18 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 310 of file [utils.c](#).

```

00311 {
00312     xmlChar buffer[64];
00313     snprintf ((char *) buffer, 64, "%.14lg", value);
00314     xmlSetProp (node, prop, buffer);
00315 }

```

4.23.2.19 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 272 of file [utils.c](#).

```
00273 {
00274     xmlChar buffer[64];
00275     snprintf ((char *) buffer, 64, "%d", value);
00276     xmlSetProp (node, prop, buffer);
00277 }
```

4.23.2.20 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 291 of file [utils.c](#).

```
00292 {
00293     xmlChar buffer[64];
00294     snprintf ((char *) buffer, 64, "%u", value);
00295     xmlSetProp (node, prop, buffer);
00296 }
```

4.24 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 Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, 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,
00060                                        double default_value,
00061                                        int *error_code);
00062 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00063 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00064                        unsigned int value);
00065 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00066 int json_object_get_int (JsonObject * object, const char *prop,
00067                        int *error_code);
00068 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00069                                   int *error_code);
00070 unsigned int json_object_get_uint_with_default (JsonObject * object,
00071                                                const char *prop,
00072                                                unsigned int default_value,
00073                                                int *error_code);
00074 double json_object_get_float (JsonObject * object, const char *prop,
00075                              int *error_code);
00076 double json_object_get_float_with_default (JsonObject * object,
00077                                           const char *prop,
00078                                           double default_value,
00079                                           int *error_code);
00080 void json_object_set_int (JsonObject * object, const char *prop, int value);
00081 void json_object_set_uint (JsonObject * object, const char *prop,
00082                          unsigned int value);
00083 void json_object_set_float (JsonObject * object, const char *prop,
00084                          double value);
00085 int cores_number ();
00086 #if HAVE_GTK
00087 void process_pending ();

```

```

00100 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00101 #endif
00102
00103 #endif

```

4.25 variable.c File Reference

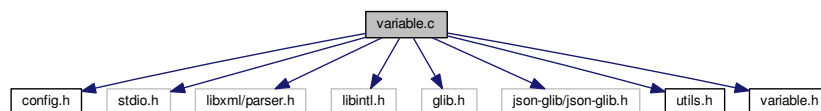
Source file to define the variable data.

```

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

```

Include dependency graph for variable.c:



Macros

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

Functions

- void [variable_new](#) ([Variable](#) *variable)
Function to create a new [Variable](#) struct.
- void [variable_free](#) ([Variable](#) *variable, unsigned int type)
Function to free the memory of a [Variable](#) struct.
- void [variable_error](#) ([Variable](#) *variable, char *message)
Function to print a message error opening an [Variable](#) struct.
- int [variable_open_xml](#) ([Variable](#) *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.
- int [variable_open_json](#) ([Variable](#) *variable, JsonNode *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.

Variables

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

4.25.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.25.2 Function Documentation

4.25.2.1 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 110 of file [variable.c](#).

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

4.25.2.2 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 87 of file [variable.c](#).

```

00088 {
00089     #if DEBUG_VARIABLE
00090         fprintf (stderr, "variable_free: start\n");
00091     #endif
00092     if (type == INPUT_TYPE_XML)
00093         xmlFree (variable->name);
00094     else
00095         g_free (variable->name);
00096     #if DEBUG_VARIABLE
00097         fprintf (stderr, "variable_free: end\n");
00098     #endif
00099 }
```

4.25.2.3 variable_new()

```

void variable_new (
    Variable * variable )
```

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

Parameters

<i>variable</i>	Variable struct.
-----------------	----------------------------------

Definition at line 67 of file [variable.c](#).

```

00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_new: start\n");
00071     #endif
00072     variable->name = NULL;
00073     #if DEBUG_VARIABLE
00074         fprintf (stderr, "variable_new: end\n");
00075     #endif
00076 }
```

4.25.2.4 variable_open_json()

```

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

Function to open the variable file.

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 direction search method.

Returns

1 on success, 0 on error.

Definition at line 303 of file [variable.c](#).

```

00305 {
00306     JsonObject *object;
00307     const char *label;
00308     int error_code;
00309     #if DEBUG_VARIABLE
00310     fprintf (stderr, "variable_open_json: start\n");
00311     #endif
00312     object = json_node_get_object (node);
00313     label = json_object_get_string_member (object, LABEL_NAME);
00314     if (!label)
00315     {
00316         variable_error (variable, _("no name"));
00317         goto exit_on_error;
00318     }
00319     variable->name = g_strdup (label);
00320     if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322         variable->rangemin
00323         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324         if (error_code)
00325         {
00326             variable_error (variable, _("bad minimum"));
00327             goto exit_on_error;
00328         }
00329         variable->rangeminabs
00330         = json_object_get_float_with_default (object,
00331         LABEL_ABSOLUTE_MINIMUM,
00332         -G_MAXDOUBLE, &error_code);
00333         if (error_code)
00334         {
00335             variable_error (variable, _("bad absolute minimum"));
00336             goto exit_on_error;
00337         }
00338         if (variable->rangemin < variable->rangeminabs)
00339         {
00340             variable_error (variable, _("minimum range not allowed"));
00341             goto exit_on_error;
00342         }
00343     }
00344     else
00345     {
00346         variable_error (variable, _("no minimum range"));
00347         goto exit_on_error;
00348     }
00349     if (json_object_get_member (object, LABEL_MAXIMUM))
00350     {
00351         variable->rangemax
00352         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00353         if (error_code)
00354         {
00355             variable_error (variable, _("bad maximum"));
00356             goto exit_on_error;
00357         }
00358         variable->rangemaxabs
00359         = json_object_get_float_with_default (object,
00360         LABEL_ABSOLUTE_MAXIMUM,
00361         G_MAXDOUBLE, &error_code);
00362         if (error_code)
00363         {
00364             variable_error (variable, _("bad absolute maximum"));
00365             goto exit_on_error;
00366         }
00367         if (variable->rangemax > variable->rangemaxabs)
00368         {

```

```

00367         variable_error (variable, _("maximum range not allowed"));
00368         goto exit_on_error;
00369     }
00370     if (variable->rangemax < variable->rangemin)
00371     {
00372         variable_error (variable, _("bad range"));
00373         goto exit_on_error;
00374     }
00375 }
00376 else
00377 {
00378     variable_error (variable, _("no maximum range"));
00379     goto exit_on_error;
00380 }
00381 variable->precision
00382 = json_object_get_uint_with_default (object,
00383 LABEL_PRECISION,
00384                                     DEFAULT_PRECISION, &error_code);
00385 if (error_code || variable->precision >= NPRECISIONS)
00386 {
00387     variable_error (variable, _("bad precision"));
00388     goto exit_on_error;
00389 }
00390 if (algorithm == ALGORITHM_SWEEP)
00391 {
00392     if (json_object_get_member (object, LABEL_NSWEEPS))
00393     {
00394         variable->nsweeps
00395         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00396         if (error_code || !variable->nsweeps)
00397         {
00398             variable_error (variable, _("bad sweeps"));
00399             goto exit_on_error;
00400         }
00401     }
00402     else
00403     {
00404         variable_error (variable, _("no sweeps number"));
00405         goto exit_on_error;
00406     }
00407     #if DEBUG_VARIABLE
00408     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00409     #endif
00410 }
00411 if (algorithm == ALGORITHM_GENETIC)
00412 {
00413     // Obtaining bits representing each variable
00414     if (json_object_get_member (object, LABEL_NBITS))
00415     {
00416         variable->nbits
00417         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00418         if (error_code || !variable->nbits)
00419         {
00420             variable_error (variable, _("invalid bits number"));
00421             goto exit_on_error;
00422         }
00423     }
00424     else
00425     {
00426         variable_error (variable, _("no bits number"));
00427         goto exit_on_error;
00428     }
00429 }
00430 else if (nsteps)
00431 {
00432     variable->step =
00433     json_object_get_float (object, LABEL_STEP, &error_code);
00434     if (error_code || variable->step < 0.)
00435     {
00436         variable_error (variable, _("bad step size"));
00437         goto exit_on_error;
00438     }
00439 }
00440 #if DEBUG_VARIABLE
00441 fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443 return 1;
00444 exit_on_error:
00445 variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447 fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449 return 0;
00450 }

```

Here is the call graph for this function:



4.25.2.5 variable_open_xml()

```

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

Function to open the variable file.

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 direction search method.

Returns

1 on success, 0 on error.

Definition at line 136 of file [variable.c](#).

```

00138 {
00139     int error_code;
00140
00141     #if DEBUG_VARIABLE
00142     fprintf (stderr, "variable_open_xml: start\n");
00143     #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, _("no name"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153         variable->rangemin
00154             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00155                                   &error_code);
00156         if (error_code)
00157         {
00158             variable_error (variable, _("bad minimum"));
00159             goto exit_on_error;
00160         }
00161     }
00162 }
  
```

```

00160     }
00161     variable->rangeminabs = xml_node_get_float_with_default
00162     (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163     &error_code);
00164     if (error_code)
00165     {
00166         variable_error (variable, _("bad absolute minimum"));
00167         goto exit_on_error;
00168     }
00169     if (variable->rangemin < variable->rangeminabs)
00170     {
00171         variable_error (variable, _("minimum range not allowed"));
00172         goto exit_on_error;
00173     }
00174 }
00175 else
00176 {
00177     variable_error (variable, _("no minimum range"));
00178     goto exit_on_error;
00179 }
00180 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181 {
00182     variable->rangemax
00183     = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00184     &error_code);
00185     if (error_code)
00186     {
00187         variable_error (variable, _("bad maximum"));
00188         goto exit_on_error;
00189     }
00190     variable->rangemaxabs = xml_node_get_float_with_default
00191     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192     &error_code);
00193     if (error_code)
00194     {
00195         variable_error (variable, _("bad absolute maximum"));
00196         goto exit_on_error;
00197     }
00198     if (variable->rangemax > variable->rangemaxabs)
00199     {
00200         variable_error (variable, _("maximum range not allowed"));
00201         goto exit_on_error;
00202     }
00203     if (variable->rangemax < variable->rangemin)
00204     {
00205         variable_error (variable, _("bad range"));
00206         goto exit_on_error;
00207     }
00208 }
00209 else
00210 {
00211     variable_error (variable, _("no maximum range"));
00212     goto exit_on_error;
00213 }
00214 variable->precision
00215 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
00216     DEFAULT_PRECISION, &error_code);
00217 if (error_code || variable->precision >= NPRECISIONS)
00218 {
00219     variable_error (variable, _("bad precision"));
00220     goto exit_on_error;
00221 }
00222 if (algorithm == ALGORITHM_SWEEP)
00223 {
00224     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225     {
00226         variable->nsweeps
00227         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00228         &error_code);
00229         if (error_code || !variable->nsweeps)
00230         {
00231             variable_error (variable, _("bad sweeps"));
00232             goto exit_on_error;
00233         }
00234     }
00235     else
00236     {
00237         variable_error (variable, _("no sweeps number"));
00238         goto exit_on_error;
00239     }
00240 #if DEBUG_VARIABLE
00241     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243 }
00244 if (algorithm == ALGORITHM_GENETIC)
00245 {

```



```

00246     // Obtaining bits representing each variable
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248     {
00249         variable->nbits
00250         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00251                             &error_code);
00252         if (error_code || !variable->nbits)
00253         {
00254             variable_error (variable, _("invalid bits number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         variable_error (variable, _("no bits number"));
00261         goto exit_on_error;
00262     }
00263 }
00264 else if (nsteps)
00265 {
00266     variable->step
00267     =
00268     xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269     if (error_code || variable->step < 0.)
00270     {
00271         variable_error (variable, _("bad step size"));
00272         goto exit_on_error;
00273     }
00274 }
00275
00276 #if DEBUG_VARIABLE
00277 fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279 return 1;
00280 exit_on_error:
00281 variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283 fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285 return 0;
00286 }

```

Here is the call graph for this function:



4.25.3 Variable Documentation

4.25.3.1 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](#).

4.25.3.2 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](#).

4.26 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 Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "variable.h"
00041
00042 #define DEBUG_VARIABLE 0
00043
00044 const char *format[NPRECISIONS] = {
00045     "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
00046     "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
00047 };
00048
00049 const double precision[NPRECISIONS] = {
00050     1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00051     1e-12, 1e-13, 1e-14
00052 };
00053
00054 void
00055 variable_new (Variable * variable)
00056 {
00057     #if DEBUG_VARIABLE
00058
```

```

00070     fprintf (stderr, "variable_new: start\n");
00071 #endif
00072     variable->name = NULL;
00073 #if DEBUG_VARIABLE
00074     fprintf (stderr, "variable_new: end\n");
00075 #endif
00076 }
00077
00086 void
00087 variable_free (Variable * variable, unsigned int type)
00088 {
00089 #if DEBUG_VARIABLE
00090     fprintf (stderr, "variable_free: start\n");
00091 #endif
00092     if (type == INPUT_TYPE_XML)
00093         xmlFree (variable->name);
00094     else
00095         g_free (variable->name);
00096 #if DEBUG_VARIABLE
00097     fprintf (stderr, "variable_free: end\n");
00098 #endif
00099 }
00100
00109 void
00110 variable_error (Variable * variable, char *message)
00111 {
00112     char buffer[64];
00113     if (!variable->name)
00114         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name,
00117                 message);
00118     error_message = g_strdup (buffer);
00119 }
00120
00135 int
00136 variable_open_xml (Variable * variable, xmlNode * node,
00137                  unsigned int algorithm, unsigned int nsteps)
00138 {
00139     int error_code;
00140
00141 #if DEBUG_VARIABLE
00142     fprintf (stderr, "variable_open_xml: start\n");
00143 #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, _("no name"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153         variable->rangemin
00154             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00155                                   &error_code);
00156         if (error_code)
00157         {
00158             variable_error (variable, _("bad minimum"));
00159             goto exit_on_error;
00160         }
00161         variable->rangeminabs = xml_node_get_float_with_default
00162             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163             &error_code);
00164         if (error_code)
00165         {
00166             variable_error (variable, _("bad absolute minimum"));
00167             goto exit_on_error;
00168         }
00169         if (variable->rangemin < variable->rangeminabs)
00170         {
00171             variable_error (variable, _("minimum range not allowed"));
00172             goto exit_on_error;
00173         }
00174     }
00175     else
00176     {
00177         variable_error (variable, _("no minimum range"));
00178         goto exit_on_error;
00179     }
00180     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181     {
00182         variable->rangemax
00183             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00184                                   &error_code);
00185         if (error_code)
00186         {

```

```

00187         variable_error (variable, _("bad maximum"));
00188         goto exit_on_error;
00189     }
00190     variable->rangemaxabs = xml_node_get_float_with_default
00191     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192      &error_code);
00193     if (error_code)
00194     {
00195         variable_error (variable, _("bad absolute maximum"));
00196         goto exit_on_error;
00197     }
00198     if (variable->rangemax > variable->rangemaxabs)
00199     {
00200         variable_error (variable, _("maximum range not allowed"));
00201         goto exit_on_error;
00202     }
00203     if (variable->rangemax < variable->rangemin)
00204     {
00205         variable_error (variable, _("bad range"));
00206         goto exit_on_error;
00207     }
00208 }
00209 else
00210 {
00211     variable_error (variable, _("no maximum range"));
00212     goto exit_on_error;
00213 }
00214 variable->precision
00215 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                DEFAULT_PRECISION, &error_code);
00216 if (error_code || variable->precision >= NPRECISIONS)
00217 {
00218     variable_error (variable, _("bad precision"));
00219     goto exit_on_error;
00220 }
00221 if (algorithm == ALGORITHM_SWEEP)
00222 {
00223     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00224     {
00225         variable->nsweeps
00226         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00227                             &error_code);
00228         if (error_code || !variable->nsweeps)
00229         {
00230             variable_error (variable, _("bad sweeps"));
00231             goto exit_on_error;
00232         }
00233     }
00234     else
00235     {
00236         variable_error (variable, _("no sweeps number"));
00237         goto exit_on_error;
00238     }
00239 }
00240 #if DEBUG_VARIABLE
00241 fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243 }
00244 if (algorithm == ALGORITHM_GENETIC)
00245 {
00246     // Obtaining bits representing each variable
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248     {
00249         variable->nbits
00250         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00251                             &error_code);
00252         if (error_code || !variable->nbits)
00253         {
00254             variable_error (variable, _("invalid bits number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         variable_error (variable, _("no bits number"));
00261         goto exit_on_error;
00262     }
00263 }
00264 else if (nsteps)
00265 {
00266     variable->step
00267     =
00268     xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269     if (error_code || variable->step < 0.)
00270     {
00271         variable_error (variable, _("bad step size"));
00272         goto exit_on_error;

```

```

00273     }
00274 }
00275
00276 #if DEBUG_VARIABLE
00277 fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279 return 1;
00280 exit_on_error:
00281     variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283 fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285 return 0;
00286 }
00287
00302 int
00303 variable_open_json (Variable * variable, JsonNode * node,
00304                     unsigned int algorithm, unsigned int nsteps)
00305 {
00306     JsonObject *object;
00307     const char *label;
00308     int error_code;
00309     #if DEBUG_VARIABLE
00310     fprintf (stderr, "variable_open_json: start\n");
00311     #endif
00312     object = json_node_get_object (node);
00313     label = json_object_get_string_member (object, LABEL_NAME);
00314     if (!label)
00315     {
00316         variable_error (variable, _("no name"));
00317         goto exit_on_error;
00318     }
00319     variable->name = g_strdup (label);
00320     if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322         variable->rangemin
00323             = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324         if (error_code)
00325         {
00326             variable_error (variable, _("bad minimum"));
00327             goto exit_on_error;
00328         }
00329         variable->rangeminabs
00330             = json_object_get_float_with_default (object,
00331 LABEL_ABSOLUTE_MINIMUM,
00332                                                     -G_MAXDOUBLE, &error_code);
00333         if (error_code)
00334         {
00335             variable_error (variable, _("bad absolute minimum"));
00336             goto exit_on_error;
00337         }
00338         if (variable->rangemin < variable->rangeminabs)
00339         {
00340             variable_error (variable, _("minimum range not allowed"));
00341             goto exit_on_error;
00342         }
00343     }
00344     else
00345     {
00346         variable_error (variable, _("no minimum range"));
00347         goto exit_on_error;
00348     }
00349     if (json_object_get_member (object, LABEL_MAXIMUM))
00350     {
00351         variable->rangemax
00352             = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00353         if (error_code)
00354         {
00355             variable_error (variable, _("bad maximum"));
00356             goto exit_on_error;
00357         }
00358         variable->rangemaxabs
00359             = json_object_get_float_with_default (object,
00360 LABEL_ABSOLUTE_MAXIMUM,
00361                                                     G_MAXDOUBLE, &error_code);
00362         if (error_code)
00363         {
00364             variable_error (variable, _("bad absolute maximum"));
00365             goto exit_on_error;
00366         }
00367         if (variable->rangemax > variable->rangemaxabs)
00368         {
00369             variable_error (variable, _("maximum range not allowed"));
00370             goto exit_on_error;
00371         }
00372         if (variable->rangemax < variable->rangemin)
00373         {

```

```

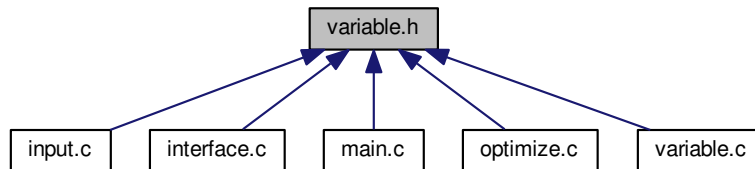
00372         variable_error (variable, _("bad range"));
00373         goto exit_on_error;
00374     }
00375 }
00376 else
00377 {
00378     variable_error (variable, _("no maximum range"));
00379     goto exit_on_error;
00380 }
00381 variable->precision
00382 = json_object_get_uint_with_default (object,
LABEL_PRECISION,
00383                                     DEFAULT_PRECISION, &error_code);
00384 if (error_code || variable->precision >= NPRECISIONS)
00385 {
00386     variable_error (variable, _("bad precision"));
00387     goto exit_on_error;
00388 }
00389 if (algorithm == ALGORITHM_SWEEP)
00390 {
00391     if (json_object_get_member (object, LABEL_NSWEEPS))
00392     {
00393         variable->nsweeps
00394         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00395         if (error_code || !variable->nsweeps)
00396         {
00397             variable_error (variable, _("bad sweeps"));
00398             goto exit_on_error;
00399         }
00400     }
00401     else
00402     {
00403         variable_error (variable, _("no sweeps number"));
00404         goto exit_on_error;
00405     }
00406 #if DEBUG_VARIABLE
00407     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00408 #endif
00409 }
00410 if (algorithm == ALGORITHM_GENETIC)
00411 {
00412     // Obtaining bits representing each variable
00413     if (json_object_get_member (object, LABEL_NBITS))
00414     {
00415         variable->nbits
00416         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00417         if (error_code || !variable->nbits)
00418         {
00419             variable_error (variable, _("invalid bits number"));
00420             goto exit_on_error;
00421         }
00422     }
00423     else
00424     {
00425         variable_error (variable, _("no bits number"));
00426         goto exit_on_error;
00427     }
00428 }
00429 else if (nsteps)
00430 {
00431     variable->step =
00432     json_object_get_float (object, LABEL_STEP, &error_code);
00433     if (error_code || variable->step < 0.)
00434     {
00435         variable_error (variable, _("bad step size"));
00436         goto exit_on_error;
00437     }
00438 }
00439 #if DEBUG_VARIABLE
00440 fprintf (stderr, "variable_open_json: end\n");
00441 #endif
00442 return 1;
00443 exit_on_error:
00444 variable_free (variable, INPUT_TYPE_JSON);
00445 #if DEBUG_VARIABLE
00446 fprintf (stderr, "variable_open_json: end\n");
00447 #endif
00448 return 0;
00449 }
00450 }

```

4.27 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 }
Enum to define the algorithms.

Functions

- void [variable_new](#) ([Variable](#) *variable)
Function to create a new [Variable](#) struct.
- void [variable_free](#) ([Variable](#) *variable, unsigned int type)
Function to free the memory of a [Variable](#) struct.
- void [variable_error](#) ([Variable](#) *variable, char *message)
Function to print a message error opening an [Variable](#) struct.
- int [variable_open_xml](#) ([Variable](#) *variable, xmlDoc *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.
- int [variable_open_json](#) ([Variable](#) *variable, cJSON *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.

Variables

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

4.27.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

4.27.2 Enumeration Type Documentation

4.27.2.1 Algorithm

enum [Algorithm](#)

Enum to define the algorithms.

Enumerator

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

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

```
00046 {  
00047     ALGORITHM_MONTE_CARLO = 0,  
00048     ALGORITHM_SWEEP = 1,  
00049     ALGORITHM_GENETIC = 2  
00050 };
```

4.27.3 Function Documentation

4.27.3.1 [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 110 of file [variable.c](#).

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

4.27.3.2 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 87 of file [variable.c](#).

```
00088 {
00089     #if DEBUG_VARIABLE
00090         fprintf (stderr, "variable_free: start\n");
00091     #endif
00092     if (type == INPUT_TYPE_XML)
00093         xmlFree (variable->name);
00094     else
00095         g_free (variable->name);
00096     #if DEBUG_VARIABLE
00097         fprintf (stderr, "variable_free: end\n");
00098     #endif
00099 }
```

4.27.3.3 variable_new()

```
void variable_new (
    Variable * variable )
```

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

Parameters

<i>variable</i>	Variable struct.
-----------------	----------------------------------

Definition at line 67 of file [variable.c](#).

```

00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_new: start\n");
00071     #endif
00072     variable->name = NULL;
00073     #if DEBUG_VARIABLE
00074         fprintf (stderr, "variable_new: end\n");
00075     #endif
00076 }
```

4.27.3.4 variable_open_json()

```

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

Function to open the variable file.

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 direction search method.

Returns

1 on success, 0 on error.

Definition at line 303 of file [variable.c](#).

```

00305 {
00306     JsonObject *object;
00307     const char *label;
00308     int error_code;
00309     #if DEBUG_VARIABLE
00310         fprintf (stderr, "variable_open_json: start\n");
00311     #endif
00312     object = json_node_get_object (node);
00313     label = json_object_get_string_member (object, LABEL_NAME);
00314     if (!label)
00315     {
00316         variable_error (variable, _("no name"));
00317         goto exit_on_error;
00318     }
00319     variable->name = g_strdup (label);
00320     if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322         variable->rangemin
```

```

00323         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324     if (error_code)
00325     {
00326         variable_error (variable, _("bad minimum"));
00327         goto exit_on_error;
00328     }
00329     variable->rangeminabs
00330     = json_object_get_float_with_default (object,
00331     LABEL_ABSOLUTE_MINIMUM,
00332                                         -G_MAXDOUBLE, &error_code);
00333     if (error_code)
00334     {
00335         variable_error (variable, _("bad absolute minimum"));
00336         goto exit_on_error;
00337     }
00338     if (variable->rangemin < variable->rangeminabs)
00339     {
00340         variable_error (variable, _("minimum range not allowed"));
00341         goto exit_on_error;
00342     }
00343     else
00344     {
00345         variable_error (variable, _("no minimum range"));
00346         goto exit_on_error;
00347     }
00348     if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350         variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352         if (error_code)
00353         {
00354             variable_error (variable, _("bad maximum"));
00355             goto exit_on_error;
00356         }
00357         variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
00359     LABEL_ABSOLUTE_MAXIMUM,
00360                                         G_MAXDOUBLE, &error_code);
00361         if (error_code)
00362         {
00363             variable_error (variable, _("bad absolute maximum"));
00364             goto exit_on_error;
00365         }
00366         if (variable->rangemax > variable->rangemaxabs)
00367         {
00368             variable_error (variable, _("maximum range not allowed"));
00369             goto exit_on_error;
00370         }
00371         if (variable->rangemax < variable->rangemin)
00372         {
00373             variable_error (variable, _("bad range"));
00374             goto exit_on_error;
00375         }
00376     }
00377     else
00378     {
00379         variable_error (variable, _("no maximum range"));
00380         goto exit_on_error;
00381     }
00382     variable->precision
00383     = json_object_get_uint_with_default (object,
00384     LABEL_PRECISION,
00385                                         DEFAULT_PRECISION, &error_code);
00386     if (error_code || variable->precision >= NPRECISIONS)
00387     {
00388         variable_error (variable, _("bad precision"));
00389         goto exit_on_error;
00390     }
00391     if (algorithm == ALGORITHM_SWEEP)
00392     {
00393         if (json_object_get_member (object, LABEL_NSWEEPS))
00394         {
00395             variable->nsweeps
00396             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00397             if (error_code || !variable->nsweeps)
00398             {
00399                 variable_error (variable, _("bad sweeps"));
00400                 goto exit_on_error;
00401             }
00402         }
00403     }
00404     else
00405     {
00406         variable_error (variable, _("no sweeps number"));
00407         goto exit_on_error;
00408     }
00409 }
00410 #if DEBUG_VARIABLE

```

```

00407     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00408 #endif
00409 }
00410 if (algorithm == ALGORITHM_GENETIC)
00411 {
00412     // Obtaining bits representing each variable
00413     if (json_object_get_member (object, LABEL_NBITS))
00414     {
00415         variable->nbits
00416         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00417         if (error_code || !variable->nbits)
00418         {
00419             variable_error (variable, _("invalid bits number"));
00420             goto exit_on_error;
00421         }
00422     }
00423     else
00424     {
00425         variable_error (variable, _("no bits number"));
00426         goto exit_on_error;
00427     }
00428 }
00429 else if (nsteps)
00430 {
00431     variable->step =
00432     json_object_get_float (object, LABEL_STEP, &error_code);
00433     if (error_code || variable->step < 0.)
00434     {
00435         variable_error (variable, _("bad step size"));
00436         goto exit_on_error;
00437     }
00438 }
00439
00440 #if DEBUG_VARIABLE
00441     fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443     return 1;
00444 exit_on_error:
00445     variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447     fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449     return 0;
00450 }

```

Here is the call graph for this function:



4.27.3.5 variable_open_xml()

```

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

```

Function to open the variable file.

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 direction search method.

Returns

1 on success, 0 on error.

Definition at line 136 of file [variable.c](#).

```

00138 {
00139     int error_code;
00140
00141     #if DEBUG_VARIABLE
00142     fprintf (stderr, "variable_open_xml: start\n");
00143     #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, _("no name"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153         variable->rangemin
00154             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00155                                   &error_code);
00156         if (error_code)
00157         {
00158             variable_error (variable, _("bad minimum"));
00159             goto exit_on_error;
00160         }
00161         variable->rangeminabs = xml_node_get_float_with_default
00162             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163             &error_code);
00164         if (error_code)
00165         {
00166             variable_error (variable, _("bad absolute minimum"));
00167             goto exit_on_error;
00168         }
00169         if (variable->rangemin < variable->rangeminabs)
00170         {
00171             variable_error (variable, _("minimum range not allowed"));
00172             goto exit_on_error;
00173         }
00174     }
00175     else
00176     {
00177         variable_error (variable, _("no minimum range"));
00178         goto exit_on_error;
00179     }
00180     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181     {
00182         variable->rangemax
00183             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00184                                   &error_code);
00185         if (error_code)
00186         {
00187             variable_error (variable, _("bad maximum"));
00188             goto exit_on_error;
00189         }
00190         variable->rangemaxabs = xml_node_get_float_with_default
00191             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192             &error_code);
00193         if (error_code)
00194         {
00195             variable_error (variable, _("bad absolute maximum"));
00196             goto exit_on_error;
00197         }
00198         if (variable->rangemax > variable->rangemaxabs)
00199         {
00200             variable_error (variable, _("maximum range not allowed"));
00201             goto exit_on_error;

```

```

00202     }
00203     if (variable->rangemax < variable->rangemin)
00204     {
00205         variable_error (variable, _("bad range"));
00206         goto exit_on_error;
00207     }
00208 }
00209 else
00210 {
00211     variable_error (variable, _("no maximum range"));
00212     goto exit_on_error;
00213 }
00214 variable->precision
00215 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                DEFAULT_PRECISION, &error_code);
00216 if (error_code || variable->precision >= NPRECISIONS)
00217 {
00218     variable_error (variable, _("bad precision"));
00219     goto exit_on_error;
00220 }
00221 }
00222 if (algorithm == ALGORITHM_SWEEP)
00223 {
00224     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225     {
00226         variable->nsweeps
00227         = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
                                &error_code);
00228         if (error_code || !variable->nsweeps)
00229         {
00230             variable_error (variable, _("bad sweeps"));
00231             goto exit_on_error;
00232         }
00233     }
00234 }
00235 else
00236 {
00237     variable_error (variable, _("no sweeps number"));
00238     goto exit_on_error;
00239 }
00240 #if DEBUG_VARIABLE
00241     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243 }
00244 if (algorithm == ALGORITHM_GENETIC)
00245 {
00246     // Obtaining bits representing each variable
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248     {
00249         variable->nbits
00250         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                &error_code);
00251         if (error_code || !variable->nbits)
00252         {
00253             variable_error (variable, _("invalid bits number"));
00254             goto exit_on_error;
00255         }
00256     }
00257 }
00258 else
00259 {
00260     variable_error (variable, _("no bits number"));
00261     goto exit_on_error;
00262 }
00263 }
00264 else if (nsteps)
00265 {
00266     variable->step
00267     =
00268     xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269     if (error_code || variable->step < 0.)
00270     {
00271         variable_error (variable, _("bad step size"));
00272         goto exit_on_error;
00273     }
00274 }
00275 }
00276 #if DEBUG_VARIABLE
00277     fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279     return 1;
00280 exit_on_error:
00281     variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283     fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285     return 0;
00286 }

```

Here is the call graph for this function:



4.28 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 2012-2016, 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 };
00041
00042 typedef struct
00043 {
00044     char *name;
00045     double rangemin;
00046     double rangemax;
00047     double rangeminabs;
00048     double rangemaxabs;
00049     double step;
00050     unsigned int precision;
00051     unsigned int nsweeps;
00052     unsigned int nbits;
00053 } Variable;
00054
00055 extern const char *format[NPRECISIONS];
00056 extern const double precision[NPRECISIONS];
00057
00058 // Public functions
00059 void variable_new (Variable * variable);
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,
  
```

```
00077             unsigned int algorithm, unsigned int nsteps);
00078 int variable_open_json (Variable * variable, JsonNode * node,
00079             unsigned int algorithm, unsigned int nsteps);
00080
00081 #endif
```


Index

Algorithm
variable.h, 278

config.h, 19
INPUT_TYPE, 22

cores_number
utils.c, 231
utils.h, 250

DirectionMethod
input.h, 69

ErrorNorm
input.h, 69

Experiment, 5
experiment.c, 24
experiment_error, 25
experiment_free, 26
experiment_new, 26
experiment_open_json, 27
experiment_open_xml, 29
template, 31

experiment.h, 35
experiment_error, 36
experiment_free, 36
experiment_new, 37
experiment_open_json, 38
experiment_open_xml, 40

experiment_error
experiment.c, 25
experiment.h, 36

experiment_free
experiment.c, 26
experiment.h, 36

experiment_new
experiment.c, 26
experiment.h, 37

experiment_open_json
experiment.c, 27
experiment.h, 38

experiment_open_xml
experiment.c, 29
experiment.h, 40

format
variable.c, 271

gtk_array_get_active
interface.h, 145
utils.c, 231
utils.h, 250

INPUT_TYPE
config.h, 22

Input, 6

input.c, 42
input_error, 43
input_open, 44
input_open_json, 45
input_open_xml, 50

input.h, 67
DirectionMethod, 69
ErrorNorm, 69
input_error, 70
input_open, 70
input_open_json, 71
input_open_xml, 77

input_error
input.c, 43
input.h, 70

input_open
input.c, 44
input.h, 70

input_open_json
input.c, 45
input.h, 71

input_open_xml
input.c, 50
input.h, 77

input_save
interface.c, 86
interface.h, 145

input_save_direction_json
interface.c, 87

input_save_direction_xml
interface.c, 88

input_save_json
interface.c, 89

input_save_xml
interface.c, 92

interface.c, 84
input_save, 86
input_save_direction_json, 87
input_save_direction_xml, 88
input_save_json, 89
input_save_xml, 92
window_get_algorithm, 94
window_get_direction, 95
window_get_norm, 96
window_new, 96
window_read, 105

- window_save, 108
 - window_template_experiment, 110
- interface.h, 142
 - gtk_array_get_active, 145
 - input_save, 145
 - window_get_algorithm, 146
 - window_get_direction, 147
 - window_get_norm, 148
 - window_new, 148
 - window_read, 157
 - window_save, 160
 - window_template_experiment, 162
- json_object_get_float
 - utils.c, 232
 - utils.h, 251
- json_object_get_float_with_default
 - utils.c, 233
 - utils.h, 251
- json_object_get_int
 - utils.c, 234
 - utils.h, 252
- json_object_get_uint
 - utils.c, 234
 - utils.h, 253
- json_object_get_uint_with_default
 - utils.c, 235
 - utils.h, 254
- json_object_set_float
 - utils.c, 236
 - utils.h, 255
- json_object_set_int
 - utils.c, 236
 - utils.h, 255
- json_object_set_uint
 - utils.c, 237
 - utils.h, 255
- main.c, 165
- Optimize, 7
 - thread_direction, 10
- optimize.c, 169
 - optimize_best, 172
 - optimize_best_direction, 173
 - optimize_direction_sequential, 173
 - optimize_direction_thread, 174
 - optimize_estimate_direction_coordinates, 175
 - optimize_estimate_direction_random, 177
 - optimize_genetic_objective, 177
 - optimize_input, 178
 - optimize_merge, 181
 - optimize_norm_euclidian, 182
 - optimize_norm_maximum, 182
 - optimize_norm_p, 183
 - optimize_norm_taxicab, 184
 - optimize_parse, 185
 - optimize_save_variables, 187
 - optimize_step_direction, 187
 - optimize_thread, 189
- optimize.h, 208
 - optimize_best, 210
 - optimize_best_direction, 211
 - optimize_direction_sequential, 212
 - optimize_direction_thread, 213
 - optimize_estimate_direction_coordinates, 214
 - optimize_estimate_direction_random, 215
 - optimize_genetic_objective, 215
 - optimize_input, 216
 - optimize_merge, 217
 - optimize_norm_euclidian, 219
 - optimize_norm_maximum, 220
 - optimize_norm_p, 221
 - optimize_norm_taxicab, 222
 - optimize_parse, 223
 - optimize_save_variables, 225
 - optimize_step_direction, 225
 - optimize_thread, 226
- optimize_best
 - optimize.c, 172
 - optimize.h, 210
- optimize_best_direction
 - optimize.c, 173
 - optimize.h, 211
- optimize_direction_sequential
 - optimize.c, 173
 - optimize.h, 212
- optimize_direction_thread
 - optimize.c, 174
 - optimize.h, 213
- optimize_estimate_direction_coordinates
 - optimize.c, 175
 - optimize.h, 214
- optimize_estimate_direction_random
 - optimize.c, 177
 - optimize.h, 215
- optimize_genetic_objective
 - optimize.c, 177
 - optimize.h, 215
- optimize_input
 - optimize.c, 178
 - optimize.h, 216
- optimize_merge
 - optimize.c, 181
 - optimize.h, 217
- optimize_norm_euclidian
 - optimize.c, 182
 - optimize.h, 219
- optimize_norm_maximum
 - optimize.c, 182
 - optimize.h, 220
- optimize_norm_p
 - optimize.c, 183
 - optimize.h, 221
- optimize_norm_taxicab
 - optimize.c, 184
 - optimize.h, 222

- optimize_parse
 - optimize.c, [185](#)
 - optimize.h, [223](#)
- optimize_save_variables
 - optimize.c, [187](#)
 - optimize.h, [225](#)
- optimize_step_direction
 - optimize.c, [187](#)
 - optimize.h, [225](#)
- optimize_thread
 - optimize.c, [189](#)
 - optimize.h, [226](#)
- Options, [11](#)
- ParallelData, [11](#)
- precision
 - variable.c, [271](#)
- Running, [12](#)
- show_error
 - utils.c, [237](#)
 - utils.h, [256](#)
- show_message
 - utils.c, [238](#)
 - utils.h, [257](#)
- template
 - experiment.c, [31](#)
- thread_direction
 - Optimize, [10](#)
- utils.c, [229](#)
 - cores_number, [231](#)
 - gtk_array_get_active, [231](#)
 - json_object_get_float, [232](#)
 - json_object_get_float_with_default, [233](#)
 - json_object_get_int, [234](#)
 - json_object_get_uint, [234](#)
 - json_object_get_uint_with_default, [235](#)
 - json_object_set_float, [236](#)
 - json_object_set_int, [236](#)
 - json_object_set_uint, [237](#)
 - show_error, [237](#)
 - show_message, [238](#)
 - xml_node_get_float, [239](#)
 - xml_node_get_float_with_default, [239](#)
 - xml_node_get_int, [240](#)
 - xml_node_get_uint, [241](#)
 - xml_node_get_uint_with_default, [242](#)
 - xml_node_set_float, [243](#)
 - xml_node_set_int, [243](#)
 - xml_node_set_uint, [243](#)
- utils.h, [248](#)
 - cores_number, [250](#)
 - gtk_array_get_active, [250](#)
 - json_object_get_float, [251](#)
 - json_object_get_float_with_default, [251](#)
 - json_object_get_int, [252](#)
 - json_object_get_uint, [253](#)
 - json_object_get_uint_with_default, [254](#)
 - json_object_set_float, [255](#)
 - json_object_set_int, [255](#)
 - json_object_set_uint, [255](#)
 - show_error, [256](#)
 - show_message, [257](#)
 - xml_node_get_float, [257](#)
 - xml_node_get_float_with_default, [258](#)
 - xml_node_get_int, [259](#)
 - xml_node_get_uint, [259](#)
 - xml_node_get_uint_with_default, [260](#)
 - xml_node_set_float, [261](#)
 - xml_node_set_int, [261](#)
 - xml_node_set_uint, [262](#)
- Variable, [12](#)
- variable.c, [264](#)
 - format, [271](#)
 - precision, [271](#)
 - variable_error, [265](#)
 - variable_free, [265](#)
 - variable_new, [266](#)
 - variable_open_json, [266](#)
 - variable_open_xml, [269](#)
- variable.h, [277](#)
 - Algorithm, [278](#)
 - variable_error, [278](#)
 - variable_free, [279](#)
 - variable_new, [279](#)
 - variable_open_json, [280](#)
 - variable_open_xml, [282](#)
- variable_error
 - variable.c, [265](#)
 - variable.h, [278](#)
- variable_free
 - variable.c, [265](#)
 - variable.h, [279](#)
- variable_new
 - variable.c, [266](#)
 - variable.h, [279](#)
- variable_open_json
 - variable.c, [266](#)
 - variable.h, [280](#)
- variable_open_xml
 - variable.c, [269](#)
 - variable.h, [282](#)
- Window, [13](#)
- window_get_algorithm
 - interface.c, [94](#)
 - interface.h, [146](#)
- window_get_direction
 - interface.c, [95](#)
 - interface.h, [147](#)
- window_get_norm
 - interface.c, [96](#)
 - interface.h, [148](#)
- window_new

- interface.c, [96](#)
 - interface.h, [148](#)
- window_read
 - interface.c, [105](#)
 - interface.h, [157](#)
- window_save
 - interface.c, [108](#)
 - interface.h, [160](#)
- window_template_experiment
 - interface.c, [110](#)
 - interface.h, [162](#)
- xml_node_get_float
 - utils.c, [239](#)
 - utils.h, [257](#)
- xml_node_get_float_with_default
 - utils.c, [239](#)
 - utils.h, [258](#)
- xml_node_get_int
 - utils.c, [240](#)
 - utils.h, [259](#)
- xml_node_get_uint
 - utils.c, [241](#)
 - utils.h, [259](#)
- xml_node_get_uint_with_default
 - utils.c, [242](#)
 - utils.h, [260](#)
- xml_node_set_float
 - utils.c, [243](#)
 - utils.h, [261](#)
- xml_node_set_int
 - utils.c, [243](#)
 - utils.h, [261](#)
- xml_node_set_uint
 - utils.c, [243](#)
 - utils.h, [262](#)