

MPCOTool

3.4.4

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	stencil	31
4.4	experiment.c	31
4.5	experiment.h File Reference	35
4.5.1	Detailed Description	36
4.5.2	Function Documentation	36
4.5.2.1	experiment_error()	36
4.5.2.2	experiment_free()	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	68
4.9.1	Detailed Description	69
4.9.2	Enumeration Type Documentation	69
4.9.2.1	DirectionMethod	69
4.9.2.2	ErrorNorm	70
4.9.3	Function Documentation	70
4.9.3.1	input_error()	70
4.9.3.2	input_open()	71
4.9.3.3	input_open_json()	73
4.9.3.4	input_open_xml()	78
4.10	input.h	84
4.11	interface.c File Reference	85
4.11.1	Detailed Description	88
4.11.2	Function Documentation	88
4.11.2.1	input_save()	88
4.11.2.2	input_save_direction_json()	89
4.11.2.3	input_save_direction_xml()	90
4.11.2.4	input_save_json()	91
4.11.2.5	input_save_xml()	94
4.11.2.6	window_get_algorithm()	96
4.11.2.7	window_get_direction()	97
4.11.2.8	window_get_norm()	98
4.11.2.9	window_new()	98
4.11.2.10	window_read()	107
4.11.2.11	window_save()	109
4.11.2.12	window_template_experiment()	111

4.12	interface.c	112
4.13	interface.h File Reference	143
4.13.1	Detailed Description	146
4.13.2	Function Documentation	146
4.13.2.1	gtk_array_get_active()	146
4.13.2.2	input_save()	147
4.13.2.3	window_get_algorithm()	148
4.13.2.4	window_get_direction()	149
4.13.2.5	window_get_norm()	149
4.13.2.6	window_new()	150
4.13.2.7	window_read()	159
4.13.2.8	window_save()	161
4.13.2.9	window_template_experiment()	163
4.14	interface.h	164
4.15	main.c File Reference	166
4.15.1	Detailed Description	167
4.16	main.c	167
4.17	optimize.c File Reference	168
4.17.1	Detailed Description	171
4.17.2	Function Documentation	171
4.17.2.1	optimize_best()	171
4.17.2.2	optimize_best_direction()	172
4.17.2.3	optimize_direction_sequential()	172
4.17.2.4	optimize_direction_thread()	173
4.17.2.5	optimize_estimate_direction_coordinates()	174
4.17.2.6	optimize_estimate_direction_random()	175
4.17.2.7	optimize_genetic_objective()	175
4.17.2.8	optimize_input()	176
4.17.2.9	optimize_merge()	178
4.17.2.10	optimize_norm_euclidian()	179

4.17.2.11 optimize_norm_maximum()	180
4.17.2.12 optimize_norm_p()	180
4.17.2.13 optimize_norm_taxicab()	181
4.17.2.14 optimize_parse()	182
4.17.2.15 optimize_save_variables()	184
4.17.2.16 optimize_step_direction()	185
4.17.2.17 optimize_thread()	186
4.18 optimize.c	187
4.19 optimize.h File Reference	204
4.19.1 Detailed Description	207
4.19.2 Function Documentation	207
4.19.2.1 optimize_best()	207
4.19.2.2 optimize_best_direction()	208
4.19.2.3 optimize_direction_sequential()	208
4.19.2.4 optimize_direction_thread()	209
4.19.2.5 optimize_estimate_direction_coordinates()	210
4.19.2.6 optimize_estimate_direction_random()	211
4.19.2.7 optimize_genetic_objective()	211
4.19.2.8 optimize_input()	212
4.19.2.9 optimize_merge()	214
4.19.2.10 optimize_norm_euclidian()	215
4.19.2.11 optimize_norm_maximum()	216
4.19.2.12 optimize_norm_p()	216
4.19.2.13 optimize_norm_taxicab()	217
4.19.2.14 optimize_parse()	218
4.19.2.15 optimize_save_variables()	220
4.19.2.16 optimize_step_direction()	221
4.19.2.17 optimize_thread()	222
4.20 optimize.h	223
4.21 utils.c File Reference	224

4.21.1 Detailed Description	226
4.21.2 Function Documentation	226
4.21.2.1 cores_number()	226
4.21.2.2 gtk_array_get_active()	227
4.21.2.3 json_object_get_float()	227
4.21.2.4 json_object_get_float_with_default()	228
4.21.2.5 json_object_get_int()	229
4.21.2.6 json_object_get_uint()	230
4.21.2.7 json_object_get_uint_with_default()	230
4.21.2.8 json_object_set_float()	231
4.21.2.9 json_object_set_int()	232
4.21.2.10 json_object_set_uint()	232
4.21.2.11 show_error()	233
4.21.2.12 show_message()	233
4.21.2.13 xml_node_get_float()	234
4.21.2.14 xml_node_get_float_with_default()	235
4.21.2.15 xml_node_get_int()	235
4.21.2.16 xml_node_get_uint()	236
4.21.2.17 xml_node_get_uint_with_default()	237
4.21.2.18 xml_node_set_float()	238
4.21.2.19 xml_node_set_int()	238
4.21.2.20 xml_node_set_uint()	239
4.22 utils.c	239
4.23 utils.h File Reference	243
4.23.1 Detailed Description	245
4.23.2 Function Documentation	245
4.23.2.1 cores_number()	245
4.23.2.2 gtk_array_get_active()	245
4.23.2.3 json_object_get_float()	246
4.23.2.4 json_object_get_float_with_default()	247

4.23.2.5	json_object_get_int()	247
4.23.2.6	json_object_get_uint()	248
4.23.2.7	json_object_get_uint_with_default()	249
4.23.2.8	json_object_set_float()	250
4.23.2.9	json_object_set_int()	250
4.23.2.10	json_object_set_uint()	251
4.23.2.11	show_error()	251
4.23.2.12	show_message()	252
4.23.2.13	xml_node_get_float()	252
4.23.2.14	xml_node_get_float_with_default()	253
4.23.2.15	xml_node_get_int()	254
4.23.2.16	xml_node_get_uint()	255
4.23.2.17	xml_node_get_uint_with_default()	255
4.23.2.18	xml_node_set_float()	256
4.23.2.19	xml_node_set_int()	257
4.23.2.20	xml_node_set_uint()	257
4.24	utils.h	257
4.25	variable.c File Reference	259
4.25.1	Detailed Description	260
4.25.2	Function Documentation	260
4.25.2.1	variable_error()	260
4.25.2.2	variable_free()	260
4.25.2.3	variable_new()	261
4.25.2.4	variable_open_json()	261
4.25.2.5	variable_open_xml()	264
4.25.3	Variable Documentation	266
4.25.3.1	format	266
4.25.3.2	precision	267
4.26	variable.c	267
4.27	variable.h File Reference	271
4.27.1	Detailed Description	273
4.27.2	Enumeration Type Documentation	273
4.27.2.1	Algorithm	273
4.27.3	Function Documentation	273
4.27.3.1	variable_error()	273
4.27.3.2	variable_free()	274
4.27.3.3	variable_new()	274
4.27.3.4	variable_open_json()	275
4.27.3.5	variable_open_xml()	277
4.28	variable.h	280

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	68
interface.c	Source file to define the graphical interface functions	85
interface.h	Header file to define the graphical interface functions	143
main.c	Main source file	166
mpcotool.c	??
mpcotool.h	??
optimize.c	Source file to define the optimization functions	168
optimize.h	Header file to define the optimization functions	204
utils.c	Source file to define some useful functions	224
utils.h	Header file to define some useful functions	243
variable.c	Source file to define the variable data	259
variable.h	Header file to define the variable data	271

Chapter 3

Data Structure Documentation

3.1 Experiment Struct Reference

Struct to define the experiment data.

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

Data Fields

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

3.1.1 Detailed Description

Struct to define the experiment data.

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

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

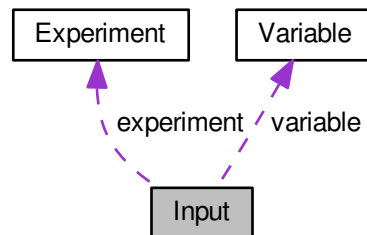
- [experiment.h](#)

3.2 Input Struct Reference

Struct to define the optimization input file.

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

Collaboration diagram for Input:



Data Fields

- [Experiment](#) * [experiment](#)
Array of experiments.
- [Variable](#) * [variable](#)
Array of variables.
- char * [result](#)
Name of the result file.
- char * [variables](#)
Name of the variables file.
- char * [simulator](#)
Name of the simulator program.
- char * [evaluator](#)
Name of the program to evaluate the objective function.
- char * [directory](#)
Working directory.
- char * [name](#)
[Input](#) data file name.
- double [tolerance](#)
Algorithm tolerance.
- double [mutation_ratio](#)
Mutation probability.
- double [reproduction_ratio](#)
Reproduction probability.
- double [adaptation_ratio](#)
Adaptation probability.
- double [relaxation](#)
Relaxation parameter.

- double [p](#)
Exponent of the P error norm.
- double [threshold](#)
Threshold to finish the optimization.
- unsigned long int [seed](#)
Seed of the pseudo-random numbers generator.
- unsigned int [nvariables](#)
Variables number.
- unsigned int [nexperiments](#)
Experiments number.
- unsigned int [nsimulations](#)
Simulations number per experiment.
- unsigned int [algorithm](#)
Algorithm type.
- unsigned int [nsteps](#)
Number of steps to do the 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:

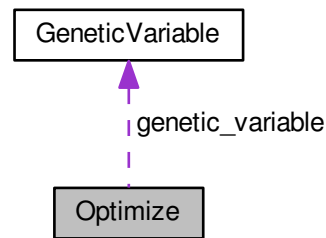
- [input.h](#)

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:

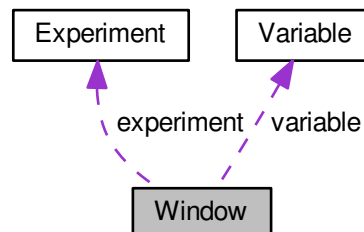
- [variable.h](#)

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

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

3.8.1 Detailed Description

Struct to define the main window.

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

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

- [interface.h](#)

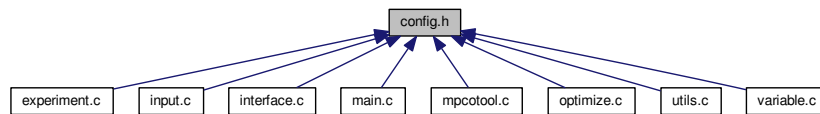
Chapter 4

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 2012-2017, 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 2012-2017, 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

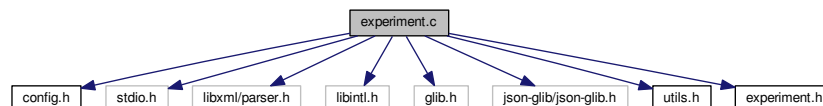
Source file to define the experiment data.

```

#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 * `stencil` [`MAX_NINPUTS`]
Array of xmlChar strings with stencil labels.

4.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2017, 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->stencil[i]);
00098         xmlFree (experiment->name);
00099     }
00100     else
00101     {
00102         for (i = 0; i < experiment->ninputs; ++i)
00103             g_free (experiment->stencil[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->stencil[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](#).

```

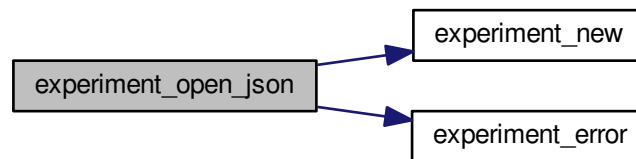
00256 {
00257     char buffer[64];
00258     JsonObject *object;
00259     const char *name;
00260     int error_code;
00261     unsigned int i;
00262
00263     #if DEBUG_EXPERIMENT
00264         fprintf (stderr, "experiment_open_json: start\n");
00265     #endif
00266
00267     // Resetting experiment 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,
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, stencil[0]);
00297 if (name)
00298 {
00299     #if DEBUG_EXPERIMENT
00300     fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00301             name, stencil[0]);
00302     #endif
00303     ++experiment->ninputs;
00304 }
00305 else
00306 {
00307     experiment_error (experiment, _("no template"));
00308     goto exit_on_error;
00309 }
00310 experiment->stencil[0] = g_strdup (name);
00311 for (i = 1; i < MAX_NINPUTS; ++i)
00312 {
00313     #if DEBUG_EXPERIMENT
00314     fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00315     #endif
00316     if (json_object_get_member (object, stencil[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, stencil[i]);
00324         #if DEBUG_EXPERIMENT
00325         fprintf (stderr,
00326                 "experiment_open_json: experiment=%s stencil%u=%s\n",
00327                 experiment->nexperiments, name, stencil[i]);
00328         #endif
00329         experiment->stencil[i] = g_strdup (name);
00330         ++experiment->ninputs;
00331     }
00332     else if (ninputs && ninputs > i)
00333     {
00334         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00335         experiment_error (experiment, buffer);
00336         goto exit_on_error;
00337     }
00338     else
00339     {
00340         break;
00341     }
00342 }
00343 #if DEBUG_EXPERIMENT
00344 fprintf (stderr, "experiment_open_json: end\n");
00345 #endif
00346 return 1;
00347 }
00348 exit_on_error:
00349 experiment_free (experiment, INPUT_TYPE_JSON);
00350 #if DEBUG_EXPERIMENT
00351 fprintf (stderr, "experiment_open_json: end\n");
00352 #endif
00353 return 0;
00354 }

```

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](#).

```

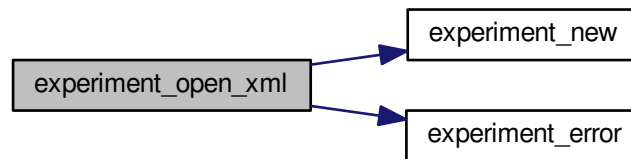
00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // 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->stencil[0]
00183     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00184     if (experiment->stencil[0])
00185     {
00186         #if DEBUG_EXPERIMENT
00187             fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00188                     experiment->name, stencil[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: stencil%u\n", i + 1);
00201         #endif
00202         if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00203         {
00204             if (ninputs && ninputs <= i)
00205             {
00206                 experiment_error (experiment, _("bad templates number"));
00207                 goto exit_on_error;
00208             }
00209             experiment->stencil[i]
00210             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00211             #if DEBUG_EXPERIMENT
00212                 fprintf (stderr,
00213                         "experiment_open_xml: experiment=%s stencil%u=%s\n",
00214                         experiment->name, experiment->nexperiments,
00215                         experiment->stencil[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 stencil

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

Initial value:

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

Array of xmlChar strings with stencil labels.

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

4.4 experiment.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

```

```

00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "experiment.h"
00041
00042 #define DEBUG_EXPERIMENT 0
00043
00044 const char *stencil[MAX_NINPUTS] = {
00045     LABEL_TEMPLATE1, LABEL_TEMPLATE2,
00046     LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00047     LABEL_TEMPLATE5, LABEL_TEMPLATE6,
00048     LABEL_TEMPLATE7, LABEL_TEMPLATE8
00049 };
00050
00051 void
00052 experiment_new (Experiment * experiment)
00053 {
00054     unsigned int i;
00055     #if DEBUG_EXPERIMENT
00056         fprintf (stderr, "experiment_new: start\n");
00057     #endif
00058     experiment->name = NULL;
00059     experiment->ninputs = 0;
00060     for (i = 0; i < MAX_NINPUTS; ++i)
00061         experiment->stencil[i] = NULL;
00062     #if DEBUG_EXPERIMENT
00063         fprintf (stderr, "input_new: end\n");
00064     #endif
00065 }
00066
00067 void
00068 experiment_free (Experiment * experiment, 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->stencil[i]);
00078         xmlFree (experiment->name);
00079     }
00080     else
00081     {
00082         for (i = 0; i < experiment->ninputs; ++i)
00083             g_free (experiment->stencil[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->stencil[0]
00183     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00184     if (experiment->stencil[0])
00185     {
00186         #if DEBUG_EXPERIMENT
00187         fprintf (stderr, "experiment_open_xml: experiment=%s stencil=%s\n",
00188         experiment->name, stencil[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: stencil%u\n", i + 1);
00201         #endif
00202         if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00203         {
00204             if (ninputs && ninputs <= i)
00205             {
00206                 experiment_error (experiment, _("bad templates number"));
00207                 goto exit_on_error;
00208             }
00209             experiment->stencil[i]
00210             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00211             #if DEBUG_EXPERIMENT
00212             fprintf (stderr,
00213             "experiment_open_xml: experiment=%s stencil%u=%s\n",
00214             experiment->nexperiments, experiment->name,
00215             experiment->stencil[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, stencil[0]);
00297     if (name)
00298     {
00299 #if DEBUG_EXPERIMENT
00300         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00301                 name, stencil[0]);
00302 #endif
00303         ++experiment->ninputs;
00304     }
00305     else
00306     {
00307         experiment_error (experiment, _("no template"));
00308         goto exit_on_error;
00309     }
00310     experiment->stencil[0] = g_strdup (name);
00311     for (i = 1; i < MAX_NINPUTS; ++i)
00312     {
00313 #if DEBUG_EXPERIMENT
00314         fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00315 #endif
00316         if (json_object_get_member (object, stencil[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, stencil[i]);
00324 #if DEBUG_EXPERIMENT
00325             fprintf (stderr,
00326                     "experiment_open_json: experiment=%s stencil%u=%s\n",
00327                     experiment->nexperiments, name, stencil[i]);
00328 #endif
00329             experiment->stencil[i] = g_strdup (name);
00330             ++experiment->ninputs;
00331         }
00332         else if (ninputs && ninputs > i)

```

```

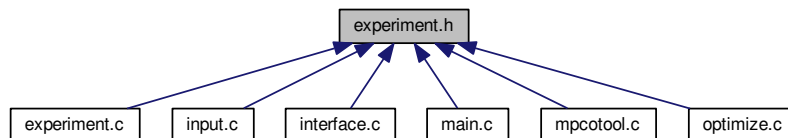
00332     {
00333         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334         experiment_error (experiment, buffer);
00335         goto exit_on_error;
00336     }
00337     else
00338         break;
00339 }
00340
00341 #if DEBUG_EXPERIMENT
00342 fprintf (stderr, "experiment_open_json: 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, 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 * stencil` [[MAX_NINPUTS](#)]
Array of `xmlChar` strings with stencil labels.

4.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2017, 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->stencil[i]);
00098         xmlFree (experiment->name);
00099     }
00100     else
00101     {
00102         for (i = 0; i < experiment->ninputs; ++i)
00103             g_free (experiment->stencil[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->stencil[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;

```

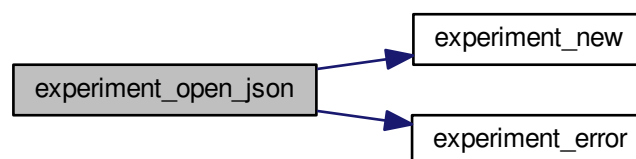


```

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, stencil[0]);
00296     if (name)
00297     {
00298     #if DEBUG_EXPERIMENT
00299         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300             name, stencil[0]);
00301     #endif
00302         ++experiment->ninputs;
00303     }
00304     else
00305     {
00306         experiment_error (experiment, _("no template"));
00307         goto exit_on_error;
00308     }
00309     experiment->stencil[0] = g_strdup (name);
00310     for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312     #if DEBUG_EXPERIMENT
00313         fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00314     #endif
00315         if (json_object_get_member (object, stencil[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, stencil[i]);
00323             #if DEBUG_EXPERIMENT
00324                 fprintf (stderr,
00325                     "experiment_open_json: experiment=%s stencil%u=%s\n",
00326                     experiment->nexperiments, name, stencil[i]);
00327             #endif
00328             experiment->stencil[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](#).

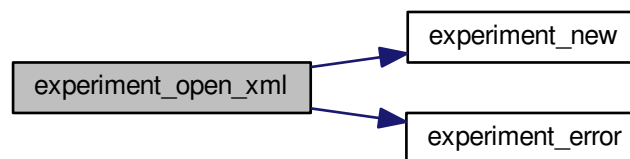
```
00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // 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->stencil[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00183     if (experiment->stencil[0])
00184     {
00185         #if DEBUG_EXPERIMENT
00186             fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
00187                     experiment->name, stencil[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: stencil%u\n", i + 1);
00200 #endif
00201         if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00202         {
00203             if (ninputs && ninputs <= i)
00204             {
00205                 experiment_error (experiment, _("bad templates number"));
00206                 goto exit_on_error;
00207             }
00208             experiment->stencil[i]
00209             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00210 #if DEBUG_EXPERIMENT
00211             fprintf (stderr,
00212                     "experiment_open_xml: experiment=%s stencil%u=%s\n",
00213                     experiment->nexperiments, experiment->name,
00214                     experiment->stencil[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 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013 1. Redistributions of source code must retain the above copyright notice,
00014 this list of conditions and the following disclaimer.
00015
00016 2. Redistributions in binary form must reproduce the above copyright notice,
00017 this list of conditions and the following disclaimer in the
00018 documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1
00034
00035 typedef struct
00036 {
00037     char *name;
00038     char *stencil[MAX_NINPUTS];
00039     double weight;
00040     unsigned int ninputs;
00041 } Experiment;
00042
00043 extern const char *stencil[MAX_NINPUTS];
00044
00045 // Public functions
00046 void experiment_new (Experiment * experiment);
00047 void experiment_free (Experiment * experiment, unsigned int type);
00048 void experiment_error (Experiment * experiment, char *message);
00049 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00050                         unsigned int ninputs);
00051 int experiment_open_json (Experiment * experiment, JsonNode * node,
00052                          unsigned int ninputs);
00053
00054 #endif

```

4.7 input.c File Reference

Source file to define the input functions.

```

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

```

Include dependency graph for input.c:



Macros

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

Functions

- `void input_new ()`
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 2012-2017, 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 958 of file [input.c](#).

```

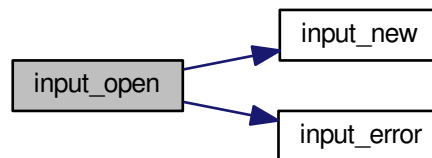
00959 {
00960     xmlDoc *doc;
00961     JsonParser *parser;
00962
00963     #if DEBUG_INPUT
00964     fprintf (stderr, "input_open: start\n");
00965     #endif
00966
00967     // Resetting input data
00968     input_new ();
00969
00970     // Opening input file
00971     #if DEBUG_INPUT
00972     fprintf (stderr, "input_open: opening the input file %s\n", filename);
00973     fprintf (stderr, "input_open: trying XML format\n");
00974     #endif
00975     doc = xmlParseFile (filename);
00976     if (!doc)
00977     {
00978         #if DEBUG_INPUT
00979         fprintf (stderr, "input_open: trying JSON format\n");
00980         #endif
00981         parser = json_parser_new ();
00982         if (!json_parser_load_from_file (parser, filename, NULL))
00983         {
00984             input_error (_("Unable to parse the input file"));
00985             goto exit_on_error;
00986         }
00987         if (!input_open_json (parser))
00988             goto exit_on_error;
00989     }
00990     else if (!input_open_xml (doc))
```

```

00991     goto exit_on_error;
00992
00993     // Getting the working directory
00994     input->directory = g_path_get_dirname (filename);
00995     input->name = g_path_get_basename (filename);
00996
00997 #if DEBUG_INPUT
00998     fprintf (stderr, "input_open: end\n");
00999 #endif
01000     return 1;
01001
01002 exit_on_error:
01003     show_error (error_message);
01004     g_free (error_message);
01005     input_free ();
01006 #if DEBUG_INPUT
01007     fprintf (stderr, "input_open: end\n");
01008 #endif
01009     return 0;
01010 }

```

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 569 of file [input.c](#).

```

00570 {
00571     JsonNode *node, *child;
00572     JsonObject *object;
00573     JsonArray *array;

```

```

00574     const char *buffer;
00575     int error_code;
00576     unsigned int i, n;
00577
00578     #if DEBUG_INPUT
00579     fprintf (stderr, "input_open_json: start\n");
00580     #endif
00581
00582     // Resetting input data
00583     input->type = INPUT_TYPE_JSON;
00584
00585     // Getting the root node
00586     #if DEBUG_INPUT
00587     fprintf (stderr, "input_open_json: getting the root node\n");
00588     #endif
00589     node = json_parser_get_root (parser);
00590     object = json_node_get_object (node);
00591
00592     // Getting result and variables file names
00593     if (!input->result)
00594     {
00595         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00596         if (!buffer)
00597             buffer = result_name;
00598         input->result = g_strdup (buffer);
00599     }
00600     else
00601         input->result = g_strdup (result_name);
00602     if (!input->variables)
00603     {
00604         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00605         if (!buffer)
00606             buffer = variables_name;
00607         input->variables = g_strdup (buffer);
00608     }
00609     else
00610         input->variables = g_strdup (variables_name);
00611
00612     // Opening simulator program name
00613     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00614     if (!buffer)
00615     {
00616         input_error (_("Bad simulator program"));
00617         goto exit_on_error;
00618     }
00619     input->simulator = g_strdup (buffer);
00620
00621     // Opening evaluator program name
00622     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00623     if (buffer)
00624         input->evaluator = g_strdup (buffer);
00625
00626     // Obtaining pseudo-random numbers generator seed
00627     input->seed
00628     = json_object_get_uint_with_default (object,
00629     LABEL_SEED,
00630     DEFAULT_RANDOM_SEED, &error_code);
00631     if (error_code)
00632     {
00633         input_error (_("Bad pseudo-random numbers generator seed"));
00634         goto exit_on_error;
00635     }
00636
00637     // Opening algorithm
00638     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00639     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00640     {
00641         input->algorithm = ALGORITHM_MONTE_CARLO;
00642
00643         // Obtaining simulations number
00644         input->nsimulations
00645         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);
00646     }
00647     else if (!strcmp (buffer, LABEL_SWEEP))
00648     {
00649         input->algorithm = ALGORITHM_SWEEP;
00650     }
00651     else if (!strcmp (buffer, LABEL_GENETIC))
00652     {
00653         input->algorithm = ALGORITHM_GENETIC;
00654
00655         // Obtaining population
00656         if (json_object_get_member (object, LABEL_NPOPULATION))

```



```

00659     {
00660         input->nsimulations
00661         = json_object_get_uint (object,
00662         LABEL_NPOPULATION, &error_code);
00663         if (error_code || input->nsimulations < 3)
00664         {
00665             input_error (_("Invalid population number"));
00666             goto exit_on_error;
00667         }
00668     else
00669     {
00670         input_error (_("No population number"));
00671         goto exit_on_error;
00672     }
00673
00674     // Obtaining generations
00675     if (json_object_get_member (object, LABEL_NGENERATIONS))
00676     {
00677         input->niterations
00678         = json_object_get_uint (object,
00679         LABEL_NGENERATIONS, &error_code);
00680         if (error_code || !input->niterations)
00681         {
00682             input_error (_("Invalid generations number"));
00683             goto exit_on_error;
00684         }
00685     else
00686     {
00687         input_error (_("No generations number"));
00688         goto exit_on_error;
00689     }
00690
00691     // Obtaining mutation probability
00692     if (json_object_get_member (object, LABEL_MUTATION))
00693     {
00694         input->mutation_ratio
00695         = json_object_get_float (object, LABEL_MUTATION, &error_code
00696 );
00697         if (error_code || input->mutation_ratio < 0.
00698         || input->mutation_ratio >= 1.)
00699         {
00700             input_error (_("Invalid mutation probability"));
00701             goto exit_on_error;
00702         }
00703     else
00704     {
00705         input_error (_("No mutation probability"));
00706         goto exit_on_error;
00707     }
00708
00709     // Obtaining reproduction probability
00710     if (json_object_get_member (object, LABEL_REPRODUCTION))
00711     {
00712         input->reproduction_ratio
00713         = json_object_get_float (object,
00714         LABEL_REPRODUCTION, &error_code);
00715         if (error_code || input->reproduction_ratio < 0.
00716         || input->reproduction_ratio >= 1.0)
00717         {
00718             input_error (_("Invalid reproduction probability"));
00719             goto exit_on_error;
00720         }
00721     else
00722     {
00723         input_error (_("No reproduction probability"));
00724         goto exit_on_error;
00725     }
00726
00727     // Obtaining adaptation probability
00728     if (json_object_get_member (object, LABEL_ADAPTATION))
00729     {
00730         input->adaptation_ratio
00731         = json_object_get_float (object,
00732         LABEL_ADAPTATION, &error_code);
00733         if (error_code || input->adaptation_ratio < 0.
00734         || input->adaptation_ratio >= 1.)
00735         {
00736             input_error (_("Invalid adaptation probability"));
00737             goto exit_on_error;
00738         }
00739     else
00740     {

```

```

00741         input_error (_("No adaptation probability"));
00742         goto exit_on_error;
00743     }
00744
00745     // Checking survivals
00746     i = input->mutation_ratio * input->nsimulations;
00747     i += input->reproduction_ratio * input->
nsimulations;
00748     i += input->adaptation_ratio * input->
nsimulations;
00749     if (i > input->nsimulations - 2)
00750     {
00751         input_error
00752         (_("No enough survival entities to reproduce the population"));
00753         goto exit_on_error;
00754     }
00755     }
00756     else
00757     {
00758         input_error (_("Unknown algorithm"));
00759         goto exit_on_error;
00760     }
00761
00762     if (input->algorithm == ALGORITHM_MONTE_CARLO
00763         || input->algorithm == ALGORITHM_SWEEP)
00764     {
00765
00766         // Obtaining iterations number
00767         input->niterations
00768         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
00769         if (error_code == 1)
00770             input->niterations = 1;
00771         else if (error_code)
00772         {
00773             input_error (_("Bad iterations number"));
00774             goto exit_on_error;
00775         }
00776
00777         // Obtaining best number
00778         input->nbest
00779         = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
00780                                             &error_code);
00781         if (error_code || !input->nbest)
00782         {
00783             input_error (_("Invalid best number"));
00784             goto exit_on_error;
00785         }
00786
00787         // Obtaining tolerance
00788         input->tolerance
00789         = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
00790                                             &error_code);
00791         if (error_code || input->tolerance < 0.)
00792         {
00793             input_error (_("Invalid tolerance"));
00794             goto exit_on_error;
00795         }
00796
00797         // Getting direction search method parameters
00798         if (json_object_get_member (object, LABEL_NSTEPS))
00799         {
00800             input->nsteps
00801             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00802             if (error_code)
00803             {
00804                 input_error (_("Invalid steps number"));
00805                 goto exit_on_error;
00806             }
00807             buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00808             if (!strcmp (buffer, LABEL_COORDINATES))
00809                 input->direction = DIRECTION_METHOD_COORDINATES;
00810             else if (!strcmp (buffer, LABEL_RANDOM))
00811             {
00812                 input->direction = DIRECTION_METHOD_RANDOM;
00813                 input->nestimates
00814                 = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00815                 if (error_code || !input->nestimates)
00816                 {
00817                     input_error (_("Invalid estimates number"));
00818                     goto exit_on_error;
00819                 }
00820             }
00821             else

```

```

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

```

```

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

```

Here is the call graph for this function:



4.7.2.4 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 DEBUG_INPUT
00175     fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00176     #endif
00177     if (!input->variables)
00178     {
00179         input->variables =
00180             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00181         if (!input->variables)
00182             input->variables =
00183                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00184     }
00185     #if DEBUG_INPUT
00186     fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00187     #endif
00188
00189     // Opening simulator program name
00190     input->simulator =
00191         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00192     if (!input->simulator)
00193     {
00194         input_error (_("Bad simulator program"));
00195         goto exit_on_error;
00196     }
00197
00198     // Opening evaluator program name
00199     input->evaluator =
00200         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00201
00202     // Obtaining pseudo-random numbers generator seed
00203     input->seed
00204         = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
00205                                         DEFAULT_RANDOM_SEED, &error_code);
00206     if (error_code)

```

```

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

```

```

00293         = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00294                               &error_code);
00295     if (error_code || input->reproduction_ratio < 0.
00296         || input->reproduction_ratio >= 1.0)
00297     {
00298         input_error (_("Invalid reproduction probability"));
00299         goto exit_on_error;
00300     }
00301 }
00302 else
00303 {
00304     input_error (_("No reproduction probability"));
00305     goto exit_on_error;
00306 }
00307
00308 // Obtaining adaptation probability
00309 if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00310 {
00311     input->adaptation_ratio
00312         = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00313                               &error_code);
00314     if (error_code || input->adaptation_ratio < 0.
00315         || input->adaptation_ratio >= 1.)
00316     {
00317         input_error (_("Invalid adaptation probability"));
00318         goto exit_on_error;
00319     }
00320 }
00321 else
00322 {
00323     input_error (_("No adaptation probability"));
00324     goto exit_on_error;
00325 }
00326
00327 // Checking survivals
00328 i = input->mutation_ratio * input->nsimulations;
00329 i += input->reproduction_ratio * input->
00330 nsimulations;
00331 i += input->adaptation_ratio * input->
00332 nsimulations;
00333 if (i > input->nsimulations - 2)
00334 {
00335     input_error
00336         (_("No enough survival entities to reproduce the population"));
00337     goto exit_on_error;
00338 }
00339 else
00340 {
00341     input_error (_("Unknown algorithm"));
00342     goto exit_on_error;
00343 }
00344 xmlFree (buffer);
00345 buffer = NULL;
00346
00347 if (input->algorithm == ALGORITHM_MONTE_CARLO
00348     || input->algorithm == ALGORITHM_SWEEP)
00349 {
00350     // Obtaining iterations number
00351     input->niterations
00352         = xml_node_get_uint (node, (const xmlChar *)
00353 LABEL_NITERATIONS,
00354                               &error_code);
00355     if (error_code == 1)
00356         input->niterations = 1;
00357     else if (error_code)
00358     {
00359         input_error (_("Bad iterations number"));
00360         goto exit_on_error;
00361     }
00362
00363     // Obtaining best number
00364     input->nbest
00365         = xml_node_get_uint_with_default (node, (const xmlChar *)
00366 LABEL_NBEST,
00367                                           1, &error_code);
00368     if (error_code || !input->nbest)
00369     {
00370         input_error (_("Invalid best number"));
00371         goto exit_on_error;
00372     }
00373     if (input->nbest > input->nsimulations)
00374     {
00375         input_error (_("Best number higher than simulations number"));
00376         goto exit_on_error;
00377     }
00378 }

```

```

00376
00377 // Obtaining tolerance
00378 input->tolerance
00379 = xml_node_get_float_with_default (node,
00380                                     (const xmlChar *) LABEL_TOLERANCE,
00381                                     0., &error_code);
00382 if (error_code || input->tolerance < 0.)
00383 {
00384     input_error (_("Invalid tolerance"));
00385     goto exit_on_error;
00386 }
00387
00388 // Getting direction search method parameters
00389 if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00390 {
00391     input->nsteps =
00392         xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00393                             &error_code);
00394     if (error_code)
00395     {
00396         input_error (_("Invalid steps number"));
00397         goto exit_on_error;
00398     }
00399 #if DEBUG_INPUT
00400     fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00401 #endif
00402     buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00403     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00404         input->direction = DIRECTION_METHOD_COORDINATES;
00405     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00406     {
00407         input->direction = DIRECTION_METHOD_RANDOM;
00408         input->nestimates
00409             = xml_node_get_uint (node, (const xmlChar *)
00410                                   LABEL_NESTIMATES,
00411                                   &error_code);
00412         if (error_code || !input->nestimates)
00413         {
00414             input_error (_("Invalid estimates number"));
00415             goto exit_on_error;
00416         }
00417     }
00418     else
00419     {
00420         input_error
00421             (_("Unknown method to estimate the direction search"));
00422         goto exit_on_error;
00423     }
00424     xmlFree (buffer);
00425     buffer = NULL;
00426     input->relaxation
00427         = xml_node_get_float_with_default (node,
00428                                             (const xmlChar *)
00429                                             LABEL_RELAXATION,
00430                                             DEFAULT_RELAXATION, &error_code);
00431     if (error_code || input->relaxation < 0. || input->
00432         relaxation > 2.)
00433     {
00434         input_error (_("Invalid relaxation parameter"));
00435         goto exit_on_error;
00436     }
00437     else
00438         input->nsteps = 0;
00439 // Obtaining the threshold
00440 input->threshold =
00441     xml_node_get_float_with_default (node, (const xmlChar *)
00442                                       LABEL_THRESHOLD,
00443                                       0., &error_code);
00444     if (error_code)
00445     {
00446         input_error (_("Invalid threshold"));
00447         goto exit_on_error;
00448     }
00449 // Reading the experimental data
00450 for (child = node->children; child; child = child->next)
00451 {
00452     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00453         break;
00454 #if DEBUG_INPUT
00455     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00456             input->nexperiments);
00457 #endif
00458     input->experiment = (Experiment *)
00459         g_realloc (input->experiment,

```



```

00460             (1 + input->nexperiments) * sizeof (
Experiment));
00461     if (!input->nexperiments)
00462     {
00463         if (!experiment_open_xml (input->experiment, child, 0))
00464             goto exit_on_error;
00465     }
00466     else
00467     {
00468         if (!experiment_open_xml (input->experiment +
input->nexperiments,
00469                                     child, input->experiment->
ninputs))
00470             goto exit_on_error;
00471     }
00472     ++input->nexperiments;
00473 #if DEBUG_INPUT
00474     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
input->nexperiments);
00475 #endif
00476     if (!input->nexperiments)
00477     {
00478         input_error (_("No optimization experiments"));
00479         goto exit_on_error;
00480     }
00481     buffer = NULL;
00482     // Reading the variables data
00483     for (; child; child = child->next)
00484     {
00485         #if DEBUG_INPUT
00486         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00487         #endif
00488         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00489         {
00490             snprintf (buffer2, 64, "%s %u: %s",
_ ("Variable"), input->nvariables + 1, _ ("bad XML node"));
00491             input_error (buffer2);
00492             goto exit_on_error;
00493         }
00494         input->variable = (Variable *)
g_realloc (input->variable,
00495             (1 + input->nvariables) * sizeof (Variable));
00496         if (!variable_open_xml (input->variable +
input->nvariables, child,
00497                                     input->algorithm, input->nsteps))
00498             goto exit_on_error;
00499         ++input->nvariables;
00500     }
00501     if (!input->nvariables)
00502     {
00503         input_error (_("No optimization variables"));
00504         goto exit_on_error;
00505     }
00506     buffer = NULL;
00507     // Obtaining the error norm
00508     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00509     {
00510         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00511         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00512             input->norm = ERROR_NORM_EUCLIDIAN;
00513         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00514             input->norm = ERROR_NORM_MAXIMUM;
00515         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00516         {
00517             input->norm = ERROR_NORM_P;
00518             input->p
= 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     }
00533     xmlFree (buffer);
00534     else
00535         input->norm = ERROR_NORM_EUCLIDIAN;

```

```

00543
00544 // Closing the XML document
00545 xmlFreeDoc (doc);
00546
00547 #if DEBUG_INPUT
00548     fprintf (stderr, "input_open_xml: end\n");
00549 #endif
00550     return 1;
00551
00552 exit_on_error:
00553     xmlFree (buffer);
00554     xmlFreeDoc (doc);
00555 #if DEBUG_INPUT
00556     fprintf (stderr, "input_open_xml: end\n");
00557 #endif
00558     return 0;
00559 }

```

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 2012-2017, 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"

```

```

00049 #include "variable.h"
00050 #include "input.h"
00051
00052 #define DEBUG_INPUT 0
00053
00054 Input input[1];
00055
00056 const char *result_name = "result";
00057 const char *variables_name = "variables";
00058
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
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node

```

```

00156 #if DEBUG_INPUT
00157     fprintf (stderr, "input_open_xml: getting the root node\n");
00158 #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (_("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173     }
00174 #if DEBUG_INPUT
00175     fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00176 #endif
00177     if (!input->variables)
00178     {
00179         input->variables =
00180             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00181         if (!input->variables)
00182             input->variables =
00183                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00184     }
00185 #if DEBUG_INPUT
00186     fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00187 #endif
00188
00189     // Opening simulator program name
00190     input->simulator =
00191         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00192     if (!input->simulator)
00193     {
00194         input_error (_("Bad simulator program"));
00195         goto exit_on_error;
00196     }
00197
00198     // Opening evaluator program name
00199     input->evaluator =
00200         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00201
00202     // Obtaining pseudo-random numbers generator seed
00203     input->seed
00204         = xml_node_get_uint_with_default (node, (const xmlChar *)
00205             LABEL_SEED,
00206             DEFAULT_RANDOM_SEED, &error_code);
00207     if (error_code)
00208     {
00209         input_error (_("Bad pseudo-random numbers generator seed"));
00210         goto exit_on_error;
00211     }
00212
00213     // Opening algorithm
00214     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00215     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00216     {
00217         input->algorithm = ALGORITHM_MONTE_CARLO;
00218
00219         // Obtaining simulations number
00220         input->nsimulations
00221             = xml_node_get_int (node, (const xmlChar *)
00222                 LABEL_NSIMULATIONS,
00223                 &error_code);
00224         if (error_code)
00225         {
00226             input_error (_("Bad simulations number"));
00227             goto exit_on_error;
00228         }
00229     }
00230     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00231         input->algorithm = ALGORITHM_SWEEP;
00232     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00233     {
00234         input->algorithm = ALGORITHM_GENETIC;
00235
00236         // Obtaining population
00237         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00238         {
00239             input->nsimulations
00240                 = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00241                     &error_code);
00242             if (error_code || input->nsimulations < 3)

```

```

00241         {
00242             input_error (_("Invalid population number"));
00243             goto exit_on_error;
00244         }
00245     }
00246     else
00247     {
00248         input_error (_("No population number"));
00249         goto exit_on_error;
00250     }
00251
00252     // Obtaining generations
00253     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00254     {
00255         input->niterations
00256             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00257                                 &error_code);
00258         if (error_code || !input->niterations)
00259         {
00260             input_error (_("Invalid generations number"));
00261             goto exit_on_error;
00262         }
00263     }
00264     else
00265     {
00266         input_error (_("No generations number"));
00267         goto exit_on_error;
00268     }
00269
00270     // Obtaining mutation probability
00271     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00272     {
00273         input->mutation_ratio
00274             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00275                                 &error_code);
00276         if (error_code || input->mutation_ratio < 0.
00277             || input->mutation_ratio >= 1.)
00278         {
00279             input_error (_("Invalid mutation probability"));
00280             goto exit_on_error;
00281         }
00282     }
00283     else
00284     {
00285         input_error (_("No mutation probability"));
00286         goto exit_on_error;
00287     }
00288
00289     // Obtaining reproduction probability
00290     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00291     {
00292         input->reproduction_ratio
00293             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00294                                 &error_code);
00295         if (error_code || input->reproduction_ratio < 0.
00296             || input->reproduction_ratio >= 1.0)
00297         {
00298             input_error (_("Invalid reproduction probability"));
00299             goto exit_on_error;
00300         }
00301     }
00302     else
00303     {
00304         input_error (_("No reproduction probability"));
00305         goto exit_on_error;
00306     }
00307
00308     // Obtaining adaptation probability
00309     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00310     {
00311         input->adaptation_ratio
00312             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00313                                 &error_code);
00314         if (error_code || input->adaptation_ratio < 0.
00315             || input->adaptation_ratio >= 1.)
00316         {
00317             input_error (_("Invalid adaptation probability"));
00318             goto exit_on_error;
00319         }
00320     }
00321     else
00322     {
00323         input_error (_("No adaptation probability"));
00324         goto exit_on_error;
00325     }
00326
00327     // Checking survivals

```

```

00328     i = input->mutation_ratio * input->nsimulations;
00329     i += input->reproduction_ratio * input->nsimulations;
00330     i += input->adaptation_ratio * input->nsimulations;
00331     if (i > input->nsimulations - 2)
00332     {
00333         input_error
00334         (_("No enough survival entities to reproduce the population"));
00335         goto exit_on_error;
00336     }
00337 }
00338 else
00339 {
00340     input_error (_("Unknown algorithm"));
00341     goto exit_on_error;
00342 }
00343 xmlFree (buffer);
00344 buffer = NULL;
00345
00346 if (input->algorithm == ALGORITHM_MONTE_CARLO
00347     || input->algorithm == ALGORITHM_SWEEP)
00348 {
00349     // Obtaining iterations number
00350     input->niterations
00351     = xml_node_get_uint (node, (const xmlChar *)
00352 LABEL_NITERATIONS,
00353                         &error_code);
00354     if (error_code == 1)
00355         input->niterations = 1;
00356     else if (error_code)
00357     {
00358         input_error (_("Bad iterations number"));
00359         goto exit_on_error;
00360     }
00361
00362     // Obtaining best number
00363     input->nbest
00364     = xml_node_get_uint_with_default (node, (const xmlChar *)
00365 LABEL_NBEST,
00366                                     1, &error_code);
00367     if (error_code || !input->nbest)
00368     {
00369         input_error (_("Invalid best number"));
00370         goto exit_on_error;
00371     }
00372     if (input->nbest > input->nsimulations)
00373     {
00374         input_error (_("Best number higher than simulations number"));
00375         goto exit_on_error;
00376     }
00377
00378     // Obtaining tolerance
00379     input->tolerance
00380     = xml_node_get_float_with_default (node,
00381                                       (const xmlChar *) LABEL_TOLERANCE,
00382                                       0., &error_code);
00383     if (error_code || input->tolerance < 0.)
00384     {
00385         input_error (_("Invalid tolerance"));
00386         goto exit_on_error;
00387     }
00388
00389     // Getting direction search method parameters
00390     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00391     {
00392         input->nsteps =
00393             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00394                               &error_code);
00395         if (error_code)
00396         {
00397             input_error (_("Invalid steps number"));
00398             goto exit_on_error;
00399         }
00400     }
00401     #if DEBUG_INPUT
00402     fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00403     #endif
00404     buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00405     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00406         input->direction = DIRECTION_METHOD_COORDINATES;
00407     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00408     {
00409         input->direction = DIRECTION_METHOD_RANDOM;
00410         input->nestimates
00411         = xml_node_get_uint (node, (const xmlChar *)
00412 LABEL_NESTIMATES,
00413                             &error_code);
00414         if (error_code || !input->nestimates)

```

```

00412         {
00413             input_error (_("Invalid estimates number"));
00414             goto exit_on_error;
00415         }
00416     }
00417     else
00418     {
00419         input_error
00420         (_("Unknown method to estimate the direction search"));
00421         goto exit_on_error;
00422     }
00423     xmlFree (buffer);
00424     buffer = NULL;
00425     input->relaxation
00426     = xml_node_get_float_with_default (node,
00427                                       (const xmlChar *)
00428                                       LABEL_RELAXATION,
00429                                       DEFAULT_RELAXATION, &error_code);
00430     if (error_code || input->relaxation < 0. || input->
00431         relaxation > 2.)
00432     {
00433         input_error (_("Invalid relaxation parameter"));
00434         goto exit_on_error;
00435     }
00436     else
00437         input->nsteps = 0;
00438 }
00439 // Obtaining the threshold
00440 input->threshold =
00441     xml_node_get_float_with_default (node, (const xmlChar *)
00442     LABEL_THRESHOLD,
00443                                     0., &error_code);
00444 if (error_code)
00445 {
00446     input_error (_("Invalid threshold"));
00447     goto exit_on_error;
00448 }
00449 // Reading the experimental data
00450 for (child = node->children; child; child = child->next)
00451 {
00452     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00453         break;
00454 #if DEBUG_INPUT
00455     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00456             input->nexperiments);
00457 #endif
00458     input->experiment = (Experiment *)
00459         g_realloc (input->experiment,
00460                   (1 + input->nexperiments) * sizeof (Experiment));
00461     if (!input->nexperiments)
00462     {
00463         if (!experiment_open_xml (input->experiment, child, 0))
00464             goto exit_on_error;
00465     }
00466     else
00467     {
00468         if (!experiment_open_xml (input->experiment + input->
00469             nexperiments,
00470                                 child, input->experiment->ninputs))
00471             goto exit_on_error;
00472         ++input->nexperiments;
00473 #if DEBUG_INPUT
00474         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00475                 input->nexperiments);
00476 #endif
00477     }
00478     if (!input->nexperiments)
00479     {
00480         input_error (_("No optimization experiments"));
00481         goto exit_on_error;
00482     }
00483     buffer = NULL;
00484 }
00485 // Reading the variables data
00486 for (; child; child = child->next)
00487 {
00488     #if DEBUG_INPUT
00489         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00490     #endif
00491     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00492     {
00493         snprintf (buffer2, 64, "%s %u: %s",
00494                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00495         input_error (buffer2);

```

```

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

```



```

00589     node = json_parser_get_root (parser);
00590     object = json_node_get_object (node);
00591
00592     // Getting result and variables file names
00593     if (!input->result)
00594     {
00595         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00596         if (!buffer)
00597             buffer = result_name;
00598         input->result = g_strdup (buffer);
00599     }
00600     else
00601         input->result = g_strdup (result_name);
00602     if (!input->variables)
00603     {
00604         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00605         if (!buffer)
00606             buffer = variables_name;
00607         input->variables = g_strdup (buffer);
00608     }
00609     else
00610         input->variables = g_strdup (variables_name);
00611
00612     // Opening simulator program name
00613     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00614     if (!buffer)
00615     {
00616         input_error (_("Bad simulator program"));
00617         goto exit_on_error;
00618     }
00619     input->simulator = g_strdup (buffer);
00620
00621     // Opening evaluator program name
00622     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00623     if (buffer)
00624         input->evaluator = g_strdup (buffer);
00625
00626     // Obtaining pseudo-random numbers generator seed
00627     input->seed
00628     = json_object_get_uint_with_default (object,
00629     LABEL_SEED,
00630     DEFAULT_RANDOM_SEED, &error_code);
00631     if (error_code)
00632     {
00633         input_error (_("Bad pseudo-random numbers generator seed"));
00634         goto exit_on_error;
00635     }
00636
00637     // Opening algorithm
00638     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00639     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00640     {
00641         input->algorithm = ALGORITHM_MONTE_CARLO;
00642     }
00643     // Obtaining simulations number
00644     input->nsimulations
00645     = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);
00646     if (error_code)
00647     {
00648         input_error (_("Bad simulations number"));
00649         goto exit_on_error;
00650     }
00651     else if (!strcmp (buffer, LABEL_SWEEP))
00652         input->algorithm = ALGORITHM_SWEEP;
00653     else if (!strcmp (buffer, LABEL_GENETIC))
00654     {
00655         input->algorithm = ALGORITHM_GENETIC;
00656     }
00657
00658     // Obtaining population
00659     if (json_object_get_member (object, LABEL_NPOPULATION))
00660     {
00661         input->nsimulations
00662         = json_object_get_uint (object,
00663         LABEL_NPOPULATION, &error_code);
00664         if (error_code || input->nsimulations < 3)
00665         {
00666             input_error (_("Invalid population number"));
00667             goto exit_on_error;
00668         }
00669     }
00670     else
00671     {
00672         input_error (_("No population number"));
00673         goto exit_on_error;
00674     }

```

```

00673
00674 // Obtaining generations
00675 if (json_object_get_member (object, LABEL_NGENERATIONS))
00676 {
00677     input->niterations
00678     = json_object_get_uint (object,
00679 LABEL_NGENERATIONS, &error_code);
00679     if (error_code || !input->niterations)
00680     {
00681         input_error (_("Invalid generations number"));
00682         goto exit_on_error;
00683     }
00684 }
00685 else
00686 {
00687     input_error (_("No generations number"));
00688     goto exit_on_error;
00689 }
00690
00691 // Obtaining mutation probability
00692 if (json_object_get_member (object, LABEL_MUTATION))
00693 {
00694     input->mutation_ratio
00695     = json_object_get_float (object, LABEL_MUTATION, &error_code
00696 );
00696     if (error_code || input->mutation_ratio < 0.
00697         || input->mutation_ratio >= 1.)
00698     {
00699         input_error (_("Invalid mutation probability"));
00700         goto exit_on_error;
00701     }
00702 }
00703 else
00704 {
00705     input_error (_("No mutation probability"));
00706     goto exit_on_error;
00707 }
00708
00709 // Obtaining reproduction probability
00710 if (json_object_get_member (object, LABEL_REPRODUCTION))
00711 {
00712     input->reproduction_ratio
00713     = json_object_get_float (object,
00714 LABEL_REPRODUCTION, &error_code);
00714     if (error_code || input->reproduction_ratio < 0.
00715         || input->reproduction_ratio >= 1.0)
00716     {
00717         input_error (_("Invalid reproduction probability"));
00718         goto exit_on_error;
00719     }
00720 }
00721 else
00722 {
00723     input_error (_("No reproduction probability"));
00724     goto exit_on_error;
00725 }
00726
00727 // Obtaining adaptation probability
00728 if (json_object_get_member (object, LABEL_ADAPTATION))
00729 {
00730     input->adaptation_ratio
00731     = json_object_get_float (object,
00732 LABEL_ADAPTATION, &error_code);
00732     if (error_code || input->adaptation_ratio < 0.
00733         || input->adaptation_ratio >= 1.)
00734     {
00735         input_error (_("Invalid adaptation probability"));
00736         goto exit_on_error;
00737     }
00738 }
00739 else
00740 {
00741     input_error (_("No adaptation probability"));
00742     goto exit_on_error;
00743 }
00744
00745 // Checking survivals
00746 i = input->mutation_ratio * input->nsimulations;
00747 i += input->reproduction_ratio * input->nsimulations;
00748 i += input->adaptation_ratio * input->nsimulations;
00749 if (i > input->nsimulations - 2)
00750 {
00751     input_error
00752     (_("No enough survival entities to reproduce the population"));
00753     goto exit_on_error;
00754 }
00755 }

```

```

00756     else
00757     {
00758         input_error (_("Unknown algorithm"));
00759         goto exit_on_error;
00760     }
00761
00762     if (input->algorithm == ALGORITHM_MONTE_CARLO
00763         || input->algorithm == ALGORITHM_SWEEP)
00764     {
00765         // Obtaining iterations number
00766         input->niterations
00767         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00768 );
00769         if (error_code == 1)
00770             input->niterations = 1;
00771         else if (error_code)
00772         {
00773             input_error (_("Bad iterations number"));
00774             goto exit_on_error;
00775         }
00776
00777         // Obtaining best number
00778         input->nbest
00779         = json_object_get_uint_with_default (object,
00780 LABEL_NBEST, 1,
00781                                             &error_code);
00782         if (error_code || !input->nbest)
00783         {
00784             input_error (_("Invalid best number"));
00785             goto exit_on_error;
00786         }
00787
00788         // Obtaining tolerance
00789         input->tolerance
00790         = json_object_get_float_with_default (object,
00791 LABEL_TOLERANCE, 0.,
00792                                             &error_code);
00793         if (error_code || input->tolerance < 0.)
00794         {
00795             input_error (_("Invalid tolerance"));
00796             goto exit_on_error;
00797         }
00798
00799         // Getting direction search method parameters
00800         if (json_object_get_member (object, LABEL_NSTEPS))
00801         {
00802             input->nsteps
00803             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00804             if (error_code)
00805             {
00806                 input_error (_("Invalid steps number"));
00807                 goto exit_on_error;
00808             }
00809             buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00810             if (!strcmp (buffer, LABEL_COORDINATES))
00811                 input->direction = DIRECTION_METHOD_COORDINATES;
00812             else if (!strcmp (buffer, LABEL_RANDOM))
00813             {
00814                 input->direction = DIRECTION_METHOD_RANDOM;
00815                 input->nestimates
00816                 = json_object_get_uint (object,
00817 LABEL_NESTIMATES, &error_code);
00818                 if (error_code || !input->nestimates)
00819                 {
00820                     input_error (_("Invalid estimates number"));
00821                     goto exit_on_error;
00822                 }
00823             }
00824             else
00825             {
00826                 input_error
00827                 (_("Unknown method to estimate the direction search"));
00828                 goto exit_on_error;
00829             }
00830             input->relaxation
00831             = json_object_get_float_with_default (object,
00832 LABEL_RELAXATION,
00833 DEFAULT_RELAXATION,
00834                                             &error_code);
00835             if (error_code || input->relaxation < 0. || input->
00836 relaxation > 2.)
00837             {
00838                 input_error (_("Invalid relaxation parameter"));
00839                 goto exit_on_error;
00840             }
00841         }
00842     }

```

```

00837         else
00838             input->nsteps = 0;
00839     }
00840     // Obtaining the threshold
00841     input->threshold
00842     = json_object_get_float_with_default (object,
LABEL_THRESHOLD, 0.,
00843                                         &error_code);
00844     if (error_code)
00845     {
00846         input_error (_("Invalid threshold"));
00847         goto exit_on_error;
00848     }
00849
00850     // Reading the experimental data
00851     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00852     n = json_array_get_length (array);
00853     input->experiment = (Experiment *) g_malloc (n * sizeof (
Experiment));
00854     for (i = 0; i < n; ++i)
00855     {
00856 #if DEBUG_INPUT
00857         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00858                 input->nexperiments);
00859 #endif
00860         child = json_array_get_element (array, i);
00861         if (!input->nexperiments)
00862         {
00863             if (!experiment_open_json (input->experiment, child, 0))
00864                 goto exit_on_error;
00865         }
00866         else
00867         {
00868             if (!experiment_open_json (input->experiment + input->
nexperiments,
00869                                     child, input->experiment->ninputs))
00870                 goto exit_on_error;
00871         }
00872         ++input->nexperiments;
00873 #if DEBUG_INPUT
00874         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00875                 input->nexperiments);
00876 #endif
00877     }
00878     if (!input->nexperiments)
00879     {
00880         input_error (_("No optimization experiments"));
00881         goto exit_on_error;
00882     }
00883
00884     // Reading the variables data
00885     array = json_object_get_array_member (object, LABEL_VARIABLES);
00886     n = json_array_get_length (array);
00887     input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00888     for (i = 0; i < n; ++i)
00889     {
00890 #if DEBUG_INPUT
00891         fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00892 #endif
00893         child = json_array_get_element (array, i);
00894         if (!variable_open_json (input->variable + input->
nvariables, child,
00895                                 input->algorithm, input->nsteps))
00896             goto exit_on_error;
00897         ++input->nvariables;
00898     }
00899     if (!input->nvariables)
00900     {
00901         input_error (_("No optimization variables"));
00902         goto exit_on_error;
00903     }
00904
00905     // Obtaining the error norm
00906     if (json_object_get_member (object, LABEL_NORM))
00907     {
00908         buffer = json_object_get_string_member (object, LABEL_NORM);
00909         if (!strcmp (buffer, LABEL_EUCLIDIAN))
00910             input->norm = ERROR_NORM_EUCLIDIAN;
00911         else if (!strcmp (buffer, LABEL_MAXIMUM))
00912             input->norm = ERROR_NORM_MAXIMUM;
00913         else if (!strcmp (buffer, LABEL_P))
00914         {
00915             input->norm = ERROR_NORM_P;
00916             input->p = json_object_get_float (object,
LABEL_P, &error_code);
00917             if (!error_code)
00918             {

```

```

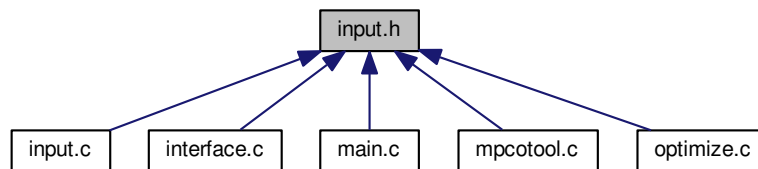
00919         input_error (_("Bad P parameter"));
00920         goto exit_on_error;
00921     }
00922 }
00923 else if (!strcmp (buffer, LABEL_TAXICAB))
00924     input->norm = ERROR_NORM_TAXICAB;
00925 else
00926 {
00927     input_error (_("Unknown error norm"));
00928     goto exit_on_error;
00929 }
00930 }
00931 else
00932     input->norm = ERROR_NORM_EUCLIDIAN;
00933
00934 // Closing the JSON document
00935 g_object_unref (parser);
00936
00937 #if DEBUG_INPUT
00938 fprintf (stderr, "input_open_json: end\n");
00939 #endif
00940 return 1;
00941
00942 exit_on_error:
00943     g_object_unref (parser);
00944 #if DEBUG_INPUT
00945     fprintf (stderr, "input_open_json: end\n");
00946 #endif
00947     return 0;
00948 }
00949
00950 int
00951 input_open (char *filename)
00952 {
00953     xmlDoc *doc;
00954     JsonParser *parser;
00955
00956 #if DEBUG_INPUT
00957     fprintf (stderr, "input_open: start\n");
00958 #endif
00959     // Resetting input data
00960     input_new ();
00961
00962     // Opening input file
00963 #if DEBUG_INPUT
00964     fprintf (stderr, "input_open: opening the input file %s\n", filename);
00965 #endif
00966     doc = xmlParseFile (filename);
00967     if (!doc)
00968     {
00969         #if DEBUG_INPUT
00970             fprintf (stderr, "input_open: trying JSON format\n");
00971         #endif
00972         parser = json_parser_new ();
00973         if (!json_parser_load_from_file (parser, filename, NULL))
00974         {
00975             input_error (_("Unable to parse the input file"));
00976             goto exit_on_error;
00977         }
00978         if (!input_open_json (parser))
00979             goto exit_on_error;
00980     }
00981     else if (!input_open_xml (doc))
00982         goto exit_on_error;
00983
00984     // Getting the working directory
00985     input->directory = g_path_get_dirname (filename);
00986     input->name = g_path_get_basename (filename);
00987
00988 #if DEBUG_INPUT
00989     fprintf (stderr, "input_open: end\n");
00990 #endif
00991     return 1;
00992
00993 exit_on_error:
00994     show_error (error_message);
00995     g_free (error_message);
00996     input_free ();
00997 #if DEBUG_INPUT
00998     fprintf (stderr, "input_open: end\n");
00999 #endif
01000     return 0;
01001 }

```

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 [ErrorNorm](#) { [ERROR_NORM_EUCLIDIAN](#) = 0, [ERROR_NORM_MAXIMUM](#) = 1, [ERROR_NORM_P](#) = 2, [ERROR_NORM_TAXICAB](#) = 3 }
- Enum to define the methods to estimate the direction search.*
- 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 2012-2017, 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 $.
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.

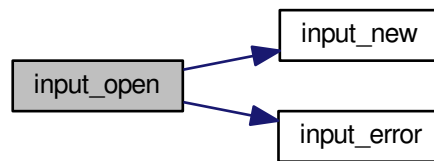
Definition at line 958 of file [input.c](#).

```

00959 {
00960     xmlDoc *doc;
00961     JsonParser *parser;
00962
00963     #if DEBUG_INPUT
00964     fprintf (stderr, "input_open: start\n");
00965     #endif
00966
00967     // Resetting input data
00968     input_new ();
00969
00970     // Opening input file
00971     #if DEBUG_INPUT
00972     fprintf (stderr, "input_open: opening the input file %s\n", filename);
00973     fprintf (stderr, "input_open: trying XML format\n");
00974     #endif
00975     doc = xmlParseFile (filename);
00976     if (!doc)
00977     {
00978         #if DEBUG_INPUT
00979         fprintf (stderr, "input_open: trying JSON format\n");
00980         #endif
00981         parser = json_parser_new ();
00982         if (!json_parser_load_from_file (parser, filename, NULL))
00983         {
00984             input_error (_("Unable to parse the input file"));
00985             goto exit_on_error;
00986         }
00987         if (!input_open_json (parser))
00988             goto exit_on_error;
00989     }
00990     else if (!input_open_xml (doc))
00991         goto exit_on_error;
00992
00993     // Getting the working directory
00994     input->directory = g_path_get_dirname (filename);
00995     input->name = g_path_get_basename (filename);
00996
00997     #if DEBUG_INPUT
00998     fprintf (stderr, "input_open: end\n");
00999     #endif
01000     return 1;
01001
01002 exit_on_error:
01003     show_error (error_message);
01004     g_free (error_message);
01005     input_free ();
01006     #if DEBUG_INPUT
01007     fprintf (stderr, "input_open: end\n");
01008     #endif
01009     return 0;
01010 }

```

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 569 of file [input.c](#).

```

00570 {
00571     JsonNode *node, *child;
00572     JsonObject *object;
00573     JsonArray *array;
00574     const char *buffer;
00575     int error_code;
00576     unsigned int i, n;
00577
00578     #if DEBUG_INPUT
00579     fprintf (stderr, "input_open_json: start\n");
00580     #endif
00581
00582     // Resetting input data
00583     input->type = INPUT_TYPE_JSON;
00584
00585     // Getting the root node
00586     #if DEBUG_INPUT
00587     fprintf (stderr, "input_open_json: getting the root node\n");
00588     #endif
00589     node = json_parser_get_root (parser);
00590     object = json_node_get_object (node);
00591
00592     // Getting result and variables file names
00593     if (!input->result)
00594     {
00595         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00596         if (!buffer)

```

```

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

```

```

00680         {
00681             input_error (_("Invalid generations number"));
00682             goto exit_on_error;
00683         }
00684     }
00685     else
00686     {
00687         input_error (_("No generations number"));
00688         goto exit_on_error;
00689     }
00690
00691     // Obtaining mutation probability
00692     if (json_object_get_member (object, LABEL_MUTATION))
00693     {
00694         input->mutation_ratio
00695         = json_object_get_float (object, LABEL_MUTATION, &error_code
00696 );
00697         if (error_code || input->mutation_ratio < 0.
00698             || input->mutation_ratio >= 1.)
00699         {
00700             input_error (_("Invalid mutation probability"));
00701             goto exit_on_error;
00702         }
00703     }
00704     else
00705     {
00706         input_error (_("No mutation probability"));
00707         goto exit_on_error;
00708     }
00709
00710     // Obtaining reproduction probability
00711     if (json_object_get_member (object, LABEL_REPRODUCTION))
00712     {
00713         input->reproduction_ratio
00714         = json_object_get_float (object,
00715 LABEL_REPRODUCTION, &error_code);
00716         if (error_code || input->reproduction_ratio < 0.
00717             || input->reproduction_ratio >= 1.0)
00718         {
00719             input_error (_("Invalid reproduction probability"));
00720             goto exit_on_error;
00721         }
00722     }
00723     else
00724     {
00725         input_error (_("No reproduction probability"));
00726         goto exit_on_error;
00727     }
00728
00729     // Obtaining adaptation probability
00730     if (json_object_get_member (object, LABEL_ADAPTATION))
00731     {
00732         input->adaptation_ratio
00733         = json_object_get_float (object,
00734 LABEL_ADAPTATION, &error_code);
00735         if (error_code || input->adaptation_ratio < 0.
00736             || input->adaptation_ratio >= 1.)
00737         {
00738             input_error (_("Invalid adaptation probability"));
00739             goto exit_on_error;
00740         }
00741     }
00742     else
00743     {
00744         input_error (_("No adaptation probability"));
00745         goto exit_on_error;
00746     }
00747
00748     // Checking survivals
00749     i = input->mutation_ratio * input->nsimulations;
00750     i += input->reproduction_ratio * input->
00751 nsimulations;
00752     i += input->adaptation_ratio * input->
00753 nsimulations;
00754     if (i > input->nsimulations - 2)
00755     {
00756         input_error
00757         (_("No enough survival entities to reproduce the population"));
00758         goto exit_on_error;
00759     }
00760 }
00761 else
00762 {
00763     input_error (_("Unknown algorithm"));
00764     goto exit_on_error;
00765 }
00766 }

```

```

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

```

```

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

```

```

00921         }
00922     }
00923     else if (!strcmp (buffer, LABEL_TAXICAB))
00924         input->norm = ERROR_NORM_TAXICAB;
00925     else
00926     {
00927         input_error (_("Unknown error norm"));
00928         goto exit_on_error;
00929     }
00930 }
00931 else
00932     input->norm = ERROR_NORM_EUCLIDIAN;
00933
00934 // Closing the JSON document
00935 g_object_unref (parser);
00936
00937 #if DEBUG_INPUT
00938 fprintf (stderr, "input_open_json: end\n");
00939 #endif
00940 return 1;
00941
00942 exit_on_error:
00943     g_object_unref (parser);
00944 #if DEBUG_INPUT
00945 fprintf (stderr, "input_open_json: end\n");
00946 #endif
00947 return 0;
00948 }

```

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 DEBUG_INPUT
00175         fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00176     #endif
00177     if (!input->variables)
00178     {
00179         input->variables =
00180             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00181         if (!input->variables)
00182             input->variables =
00183                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00184     }
00185     #if DEBUG_INPUT
00186         fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00187     #endif
00188
00189     // Opening simulator program name
00190     input->simulator =
00191         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00192     if (!input->simulator)
00193     {
00194         input_error (_("Bad simulator program"));
00195         goto exit_on_error;
00196     }
00197
00198     // Opening evaluator program name
00199     input->evaluator =
00200         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00201
00202     // Obtaining pseudo-random numbers generator seed
00203     input->seed
00204         = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
00205                                         DEFAULT_RANDOM_SEED, &error_code);
00206     if (error_code)
00207     {
00208         input_error (_("Bad pseudo-random numbers generator seed"));
00209         goto exit_on_error;
00210     }
00211
00212     // Opening algorithm
00213     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00214     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00215     {
00216         input->algorithm = ALGORITHM_MONTE_CARLO;
00217
00218         // Obtaining simulations number
00219         input->nsimulations
00220             = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
00221                               &error_code);
00222         if (error_code)
00223         {
00224             input_error (_("Bad simulations number"));

```

```

00225         goto exit_on_error;
00226     }
00227 }
00228 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00229     input->algorithm = ALGORITHM_SWEEP;
00230 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00231 {
00232     input->algorithm = ALGORITHM_GENETIC;
00233
00234     // Obtaining population
00235     if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00236     {
00237         input->nsimulations
00238             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00239                                 &error_code);
00240         if (error_code || input->nsimulations < 3)
00241         {
00242             input_error (_("Invalid population number"));
00243             goto exit_on_error;
00244         }
00245     }
00246     else
00247     {
00248         input_error (_("No population number"));
00249         goto exit_on_error;
00250     }
00251
00252     // Obtaining generations
00253     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00254     {
00255         input->niterations
00256             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00257                                 &error_code);
00258         if (error_code || !input->niterations)
00259         {
00260             input_error (_("Invalid generations number"));
00261             goto exit_on_error;
00262         }
00263     }
00264     else
00265     {
00266         input_error (_("No generations number"));
00267         goto exit_on_error;
00268     }
00269
00270     // Obtaining mutation probability
00271     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00272     {
00273         input->mutation_ratio
00274             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00275                                  &error_code);
00276         if (error_code || input->mutation_ratio < 0.
00277             || input->mutation_ratio >= 1.)
00278         {
00279             input_error (_("Invalid mutation probability"));
00280             goto exit_on_error;
00281         }
00282     }
00283     else
00284     {
00285         input_error (_("No mutation probability"));
00286         goto exit_on_error;
00287     }
00288
00289     // Obtaining reproduction probability
00290     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00291     {
00292         input->reproduction_ratio
00293             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00294                                  &error_code);
00295         if (error_code || input->reproduction_ratio < 0.
00296             || input->reproduction_ratio >= 1.0)
00297         {
00298             input_error (_("Invalid reproduction probability"));
00299             goto exit_on_error;
00300         }
00301     }
00302     else
00303     {
00304         input_error (_("No reproduction probability"));
00305         goto exit_on_error;
00306     }
00307
00308     // Obtaining adaptation probability
00309     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00310     {
00311         input->adaptation_ratio

```

```

00312         = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00313                               &error_code);
00314         if (error_code || input->adaptation_ratio < 0.
00315             || input->adaptation_ratio >= 1.)
00316         {
00317             input_error (_("Invalid adaptation probability"));
00318             goto exit_on_error;
00319         }
00320     }
00321     else
00322     {
00323         input_error (_("No adaptation probability"));
00324         goto exit_on_error;
00325     }
00326
00327     // Checking survivals
00328     i = input->mutation_ratio * input->nsimulations;
00329     i += input->reproduction_ratio * input->
00330 nsimulations;
00331     i += input->adaptation_ratio * input->
00332 nsimulations;
00333     if (i > input->nsimulations - 2)
00334     {
00335         input_error
00336             (_("No enough survival entities to reproduce the population"));
00337         goto exit_on_error;
00338     }
00339     else
00340     {
00341         input_error (_("Unknown algorithm"));
00342         goto exit_on_error;
00343     }
00344     xmlFree (buffer);
00345     buffer = NULL;
00346
00347     if (input->algorithm == ALGORITHM_MONTE_CARLO
00348         || input->algorithm == ALGORITHM_SWEEP)
00349     {
00350         // Obtaining iterations number
00351         input->niterations
00352             = xml_node_get_uint (node, (const xmlChar *)
00353 LABEL_NITERATIONS,
00354                               &error_code);
00355         if (error_code == 1)
00356             input->niterations = 1;
00357         else if (error_code)
00358         {
00359             input_error (_("Bad iterations number"));
00360             goto exit_on_error;
00361         }
00362
00363         // Obtaining best number
00364         input->nbest
00365             = xml_node_get_uint_with_default (node, (const xmlChar *)
00366 LABEL_NBEST,
00367                                             1, &error_code);
00368         if (error_code || !input->nbest)
00369         {
00370             input_error (_("Invalid best number"));
00371             goto exit_on_error;
00372         }
00373         if (input->nbest > input->nsimulations)
00374         {
00375             input_error (_("Best number higher than simulations number"));
00376             goto exit_on_error;
00377         }
00378
00379         // Obtaining tolerance
00380         input->tolerance
00381             = xml_node_get_float_with_default (node,
00382                                               (const xmlChar *) LABEL_TOLERANCE,
00383                                               0., &error_code);
00384         if (error_code || input->tolerance < 0.)
00385         {
00386             input_error (_("Invalid tolerance"));
00387             goto exit_on_error;
00388         }
00389
00390         // Getting direction search method parameters
00391         if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00392         {
00393             input->nsteps =
00394                 xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00395                                   &error_code);
00396             if (error_code)

```

```

00395         {
00396             input_error (_("Invalid steps number"));
00397             goto exit_on_error;
00398         }
00399 #if DEBUG_INPUT
00400     fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00401 #endif
00402     buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00403     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00404         input->direction = DIRECTION_METHOD_COORDINATES;
00405     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00406     {
00407         input->direction = DIRECTION_METHOD_RANDOM;
00408         input->nestimates
00409         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00410                             &error_code);
00411         if (error_code || !input->nestimates)
00412         {
00413             input_error (_("Invalid estimates number"));
00414             goto exit_on_error;
00415         }
00416     }
00417     else
00418     {
00419         input_error
00420         (_("Unknown method to estimate the direction search"));
00421         goto exit_on_error;
00422     }
00423     xmlFree (buffer);
00424     buffer = NULL;
00425     input->relaxation
00426     = xml_node_get_float_with_default (node,
00427                                       (const xmlChar *)
LABEL_RELAXATION,
00428                                       DEFAULT_RELAXATION, &error_code);
00429     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00430     {
00431         input_error (_("Invalid relaxation parameter"));
00432         goto exit_on_error;
00433     }
00434     }
00435     }
00436     else
00437         input->nsteps = 0;
00438     }
00439     // Obtaining the threshold
00440     input->threshold =
00441     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00442                                     0., &error_code);
00443     if (error_code)
00444     {
00445         input_error (_("Invalid threshold"));
00446         goto exit_on_error;
00447     }
00448     // Reading the experimental data
00449     for (child = node->children; child; child = child->next)
00450     {
00451         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00452             break;
00453     }
00454 #if DEBUG_INPUT
00455     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00456             input->nexperiments);
00457 #endif
00458     input->experiment = (Experiment *)
00459     g_realloc (input->experiment,
00460               (1 + input->nexperiments) * sizeof (
Experiment));
00461     if (!input->nexperiments)
00462     {
00463         if (!experiment_open_xml (input->experiment, child, 0))
00464             goto exit_on_error;
00465     }
00466     else
00467     {
00468         if (!experiment_open_xml (input->experiment +
input->nexperiments,
00469                                 child, input->experiment->
ninputs))
00470             goto exit_on_error;
00471     }
00472     ++input->nexperiments;
00473 #if DEBUG_INPUT
00474     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00475             input->nexperiments);

```

```

00476 #endif
00477     }
00478     if (!input->nexperiments)
00479     {
00480         input_error (_("No optimization experiments"));
00481         goto exit_on_error;
00482     }
00483     buffer = NULL;
00484
00485     // Reading the variables data
00486     for (; child; child = child->next)
00487     {
00488 #if DEBUG_INPUT
00489         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00490 #endif
00491         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00492         {
00493             snprintf (buffer2, 64, "%s %u: %s",
00494                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00495             input_error (buffer2);
00496             goto exit_on_error;
00497         }
00498         input->variable = (Variable *)
00499             g_realloc (input->variable,
00500                 (1 + input->nvariables) * sizeof (Variable));
00501         if (!variable_open_xml (input->variable +
00502             input->nvariables, child,
00503                 input->algorithm, input->nsteps))
00504             goto exit_on_error;
00505         ++input->nvariables;
00506     }
00507     if (!input->nvariables)
00508     {
00509         input_error (_("No optimization variables"));
00510         goto exit_on_error;
00511     }
00512     buffer = NULL;
00513
00514     // Obtaining the error norm
00515     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00516     {
00517         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00518         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00519             input->norm = ERROR_NORM_EUCLIDIAN;
00520         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00521             input->norm = ERROR_NORM_MAXIMUM;
00522         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00523         {
00524             input->norm = ERROR_NORM_P;
00525             input->p
00526                 = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00527             if (!error_code)
00528             {
00529                 input_error (_("Bad P parameter"));
00530                 goto exit_on_error;
00531             }
00532         }
00533         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00534             input->norm = ERROR_NORM_TAXICAB;
00535         else
00536         {
00537             input_error (_("Unknown error norm"));
00538             goto exit_on_error;
00539         }
00540         xmlFree (buffer);
00541     }
00542     else
00543         input->norm = ERROR_NORM_EUCLIDIAN;
00544
00545     // Closing the XML document
00546     xmlFreeDoc (doc);
00547
00548 #if DEBUG_INPUT
00549     fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551     return 1;
00552
00553 exit_on_error:
00554     xmlFree (buffer);
00555     xmlFreeDoc (doc);
00556 #if DEBUG_INPUT
00557     fprintf (stderr, "input_open_xml: end\n");
00558 #endif
00559     return 0;

```

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 2012-2017, 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;
  
```

```

00086 double relaxation;
00087 double p;
00088 double threshold;
00089 unsigned long int seed;
00091 unsigned int nvariables;
00092 unsigned int nexperiments;
00093 unsigned int nsimulations;
00094 unsigned int algorithm;
00095 unsigned int nsteps;
00097 unsigned int direction;
00098 unsigned int nestimates;
00100 unsigned int niterations;
00101 unsigned int nbest;
00102 unsigned int norm;
00103 unsigned int type;
00104 } Input;
00105
00106 extern Input input[1];
00107 extern const char *result_name;
00108 extern const char *variables_name;
00109
00110 // Public functions
00111 void input_new ();
00112 void input_free ();
00113 void input_error (char *message);
00114 int input_open_xml (xmlDoc * doc);
00115 int input_open_json (JsonParser * parser);
00116 int input_open (char *filename);
00117
00118 #endif

```

4.11 interface.c File Reference

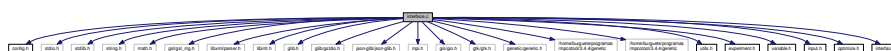
Source file to define the graphical interface functions.

```

#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 2012-2017, 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 575 of file [interface.c](#).

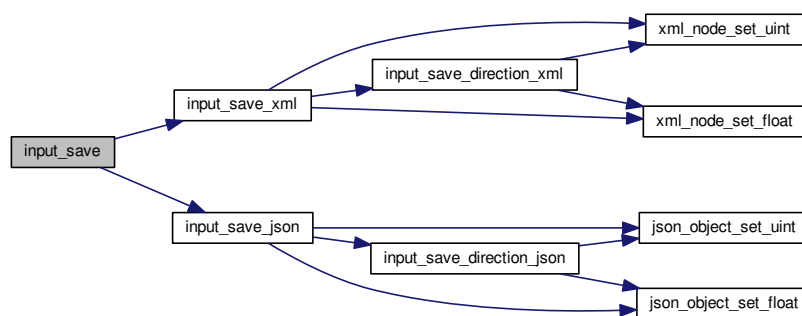
```
00576 {
00577     xmlDoc *doc;
00578     JsonGenerator *generator;
00579
00580 #if DEBUG_INTERFACE
00581     fprintf (stderr, "input_save: start\n");
00582 #endif
00583
00584     // Getting the input file directory
00585     input->name = g_path_get_basename (filename);
00586     input->directory = g_path_get_dirname (filename);
00587
00588     if (input->type == INPUT_TYPE_XML)
00589     {
00590         // Opening the input file
00591         doc = xmlNewDoc ((const xmlChar *) "1.0");
00592         input_save_xml (doc);
00593
00594         // Saving the XML file
00595         xmlSaveFormatFile (filename, doc, 1);
00596
00597         // Freeing memory
00598         xmlFreeDoc (doc);
00599     }
00600     else
00601     {
00602         // Opening the input file
```

```

00603     generator = json_generator_new ();
00604     json_generator_set_pretty (generator, TRUE);
00605     input_save_json (generator);
00606
00607     // Saving the JSON file
00608     json_generator_to_file (generator, filename, NULL);
00609
00610     // Freeing memory
00611     g_object_unref (generator);
00612 }
00613
00614 #if DEBUG_INTERFACE
00615     fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }

```

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     }
00222 }

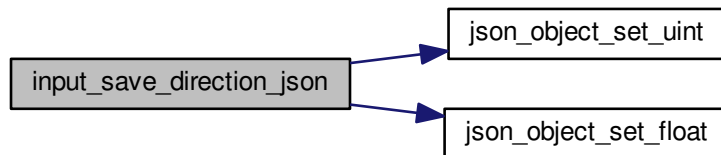
```

```

00219     switch (input->direction)
00220     {
00221         case DIRECTION_METHOD_COORDINATES:
00222             json_object_set_string_member (object, LABEL_DIRECTION,
00223                                           LABEL_COORDINATES);
00224             break;
00225         default:
00226             json_object_set_string_member (object, LABEL_DIRECTION,
00227                                           LABEL_RANDOM);
00228             json_object_set_uint (object, LABEL_NESTIMATES,
00229                                  input->nestimates);
00230     }
00231     #if DEBUG_INTERFACE
00232     fprintf (stderr, "input_save_direction_json: end\n");
00233     #endif
00234 }

```

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);
00180         if (input->relaxation != DEFAULT_RELAXATION)
00181             xml_node_set_float (node, (const xmlChar *)
00182                                 LABEL_RELAXATION,
00183                                 input->relaxation);
00184         switch (input->direction)
00185         {

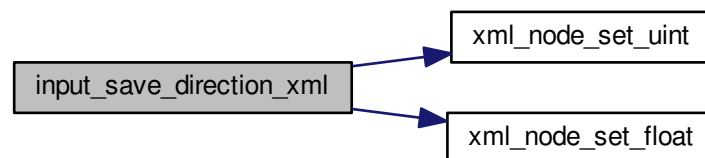
```

```

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

```

00413 {
00414     unsigned int i, j;
00415     char *buffer;
00416     JsonNode *node, *child;
00417     JsonObject *object;
00418     JsonArray *array;
00419     GFile *file, *file2;
00420
00421     #if DEBUG_INTERFACE
00422     fprintf (stderr, "input_save_json: start\n");
00423     #endif
00424
00425     // Setting root JSON node
00426     node = json_node_new (JSON_NODE_OBJECT);
00427     object = json_node_get_object (node);

```

```

00428     json_generator_set_root (generator, node);
00429
00430     // Adding properties to the root JSON node
00431     if (strcmp (input->result, result_name))
00432         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00433     if (strcmp (input->variables, variables_name))
00434         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00435     file = g_file_new_for_path (input->directory);
00436     file2 = g_file_new_for_path (input->simulator);
00437     buffer = g_file_get_relative_path (file, file2);
00438     g_object_unref (file2);
00439     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00440     g_free (buffer);
00441     if (input->evaluator)
00442     {
00443         file2 = g_file_new_for_path (input->evaluator);
00444         buffer = g_file_get_relative_path (file, file2);
00445         g_object_unref (file2);
00446         if (strlen (buffer))
00447             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00448         g_free (buffer);
00449     }
00450     if (input->seed != DEFAULT_RANDOM_SEED)
00451         json_object_set_uint (object, LABEL_SEED,
input->seed);
00452
00453     // Setting the algorithm
00454     buffer = (char *) g_slice_alloc (64);
00455     switch (input->algorithm)
00456     {
00457     case ALGORITHM_MONTE_CARLO:
00458         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00459         snprintf (buffer, 64, "%u", input->nsimulations);
00460         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00461         snprintf (buffer, 64, "%u", input->niterations);
00462         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00463         snprintf (buffer, 64, "%.3lg", input->tolerance);
00464         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00465         snprintf (buffer, 64, "%u", input->nbest);
00466         json_object_set_string_member (object, LABEL_NBEST, buffer);
00467         input_save_direction_json (node);
00468         break;
00469     case ALGORITHM_SWEEP:
00470         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00471         snprintf (buffer, 64, "%u", input->niterations);
00472         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00473         snprintf (buffer, 64, "%.3lg", input->tolerance);
00474         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00475         snprintf (buffer, 64, "%u", input->nbest);
00476         json_object_set_string_member (object, LABEL_NBEST, buffer);
00477         input_save_direction_json (node);
00478         break;
00479     default:
00480         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00481         snprintf (buffer, 64, "%u", input->nsimulations);
00482         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00483         snprintf (buffer, 64, "%u", input->niterations);
00484         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00485         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00486         json_object_set_string_member (object, LABEL_MUTATION_RATIO, buffer);
00487         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00488         json_object_set_string_member (object, LABEL_REPRODUCTION_RATIO, buffer);
00489         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00490         json_object_set_string_member (object, LABEL_ADAPTATION_RATIO, buffer);
00491         break;
00492     }
00493     g_slice_free1 (64, buffer);
00494     if (input->threshold != 0.)
00495         json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00496
00497     // Setting the experimental data
00498     array = json_array_new ();
00499     for (i = 0; i < input->nexperiments; ++i)
00500     {
00501         child = json_node_new (JSON_NODE_OBJECT);
00502         object = json_node_get_object (child);
00503         json_object_set_string_member (object, LABEL_NAME,
input->experiment[i].name);
00504         if (input->experiment[i].weight != 1.)
00505             json_object_set_float (object, LABEL_WEIGHT,
input->experiment[i].weight);
00506     }

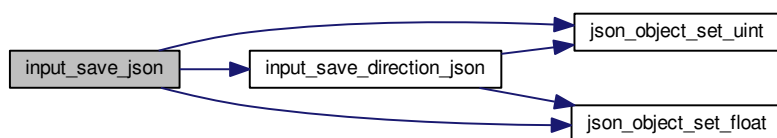
```

```

00510     for (j = 0; j < input->experiment->ninputs; ++j)
00511         json_object_set_string_member (object, stencil[j],
00512                                         input->experiment[i].
stencil[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 (object, LABEL_NAME,
00524                                     input->variable[i].name);
00525     json_object_set_float (object, LABEL_MINIMUM,
00526                             input->variable[i].rangemin);
00527     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00528         json_object_set_float (object,
LABEL_ABSOLUTE_MINIMUM,
00529                                 input->variable[i].rangeminabs);
00530     json_object_set_float (object, LABEL_MAXIMUM,
00531                             input->variable[i].rangemax);
00532     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00533         json_object_set_float (object,
LABEL_ABSOLUTE_MAXIMUM,
00534                                 input->variable[i].rangemaxabs);
00535     if (input->variable[i].precision !=
DEFAULT_PRECISION)
00536         json_object_set_uint (object, LABEL_PRECISION,
00537                                 input->variable[i].precision);
00538     if (input->algorithm == ALGORITHM_SWEEP)
00539         json_object_set_uint (object, LABEL_NSWEEPS,
00540                                 input->variable[i].nsweeps);
00541     else if (input->algorithm == ALGORITHM_GENETIC)
00542         json_object_set_uint (object, LABEL_NBITS,
input->variable[i].nbits);
00543     if (input->nsteps)
00544         json_object_set_float (object, LABEL_STEP,
input->variable[i].step);
00545     json_array_add_element (array, child);
00546 }
00547 json_object_set_array_member (object, LABEL_VARIABLES, array);
00548
00549 // Saving the error norm
00550 switch (input->norm)
00551 {
00552     case ERROR_NORM_MAXIMUM:
00553         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00554         break;
00555     case ERROR_NORM_P:
00556         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00557         json_object_set_float (object, LABEL_P, input->
p);
00558         break;
00559     case ERROR_NORM_TAXICAB:
00560         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00561 }
00562
00563 #if DEBUG_INTERFACE
00564 fprintf (stderr, "input_save_json: end\n");
00565 #endif
00566 }

```

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

```
00243 {
00244     unsigned int i, j;
00245     char *buffer;
00246     xmlNode *node, *child;
00247     GFile *file, *file2;
00248
00249     #if DEBUG_INTERFACE
00250         fprintf (stderr, "input_save_xml: start\n");
00251     #endif
00252
00253     // Setting root XML node
00254     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00255     xmlDocSetRootElement (doc, node);
00256
00257     // Adding properties to the root XML node
00258     if (xmlStrcmp
00259         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00260         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00261                     (xmlChar *) input->result);
00262     if (xmlStrcmp
00263         ((const xmlChar *) input->variables, (const xmlChar *)
00264         variables_name))
00265         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00266                     (xmlChar *) input->variables);
00267     file = g_file_new_for_path (input->directory);
00268     file2 = g_file_new_for_path (input->simulator);
00269     buffer = g_file_get_relative_path (file, file2);
00270     g_object_unref (file2);
00271     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00272     g_free (buffer);
00273     if (input->evaluator)
00274     {
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,
00285                             input->seed);
00286
00287     // Setting the algorithm
00288     buffer = (char *) g_slice_alloc (64);
00289     switch (input->algorithm)
00290     {
00291     case ALGORITHM_MONTE_CARLO:
00292         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00293                     (const xmlChar *) LABEL_MONTE_CARLO);
00294         snprintf (buffer, 64, "%u", input->nsimulations);
00295         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00296                     (xmlChar *) buffer);
00297         snprintf (buffer, 64, "%u", input->niterations);
00298         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00299                     (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%.3lg", input->tolerance);
00301         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%u", input->nbest);
00303         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304         input_save_direction_xml (node);
00305         break;
00306     case ALGORITHM_SWEEP:
```



```

00305     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00306                 (const xmlChar *) LABEL_SWEEP);
00307     snprintf (buffer, 64, "%u", input->niterations);
00308     xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00309                 (xmlChar *) buffer);
00310     snprintf (buffer, 64, "%.3lg", input->tolerance);
00311     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00312     snprintf (buffer, 64, "%u", input->nbest);
00313     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00314     input_save_direction_xml (node);
00315     break;
00316 default:
00317     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00318                 (const xmlChar *) LABEL_GENETIC);
00319     snprintf (buffer, 64, "%u", input->nsimulations);
00320     xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00321                 (xmlChar *) buffer);
00322     snprintf (buffer, 64, "%u", input->niterations);
00323     xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00324                 (xmlChar *) buffer);
00325     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00326     xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00327     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00328     xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00329                 (xmlChar *) buffer);
00330     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00331     xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00332     break;
00333 }
00334 g_slice_free1 (64, buffer);
00335 if (input->threshold != 0.)
00336     xml_node_set_float (node, (const xmlChar *)
00337 LABEL_THRESHOLD,
00338                         input->threshold);
00339 // Setting the experimental data
00340 for (i = 0; i < input->nexperiments; ++i)
00341 {
00342     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00343     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00344                 (xmlChar *) input->experiment[i].name);
00345     if (input->experiment[i].weight != 1.)
00346         xml_node_set_float (child, (const xmlChar *)
00347 LABEL_WEIGHT,
00348                             input->experiment[i].weight);
00349     for (j = 0; j < input->experiment->ninputs; ++j)
00350         xmlSetProp (child, (const xmlChar *) stencil[j],
00351                     (xmlChar *) input->experiment[i].stencil[j]);
00352 }
00353 // Setting the variables data
00354 for (i = 0; i < input->nvariables; ++i)
00355 {
00356     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00357     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00358                 (xmlChar *) input->variable[i].name);
00359     xml_node_set_float (child, (const xmlChar *)
00360 LABEL_MINIMUM,
00361                         input->variable[i].rangemin);
00362     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00363         xml_node_set_float (child, (const xmlChar *)
00364 LABEL_ABSOLUTE_MINIMUM,
00365                             input->variable[i].rangeminabs);
00366     xml_node_set_float (child, (const xmlChar *)
00367 LABEL_MAXIMUM,
00368                         input->variable[i].rangemax);
00369     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
00371 LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
00374         DEFAULT_PRECISION)
00375         xml_node_set_uint (child, (const xmlChar *)
00376 LABEL_PRECISION,
00377                             input->variable[i].precision);
00378     if (input->algorithm == ALGORITHM_SWEEP)
00379         xml_node_set_uint (child, (const xmlChar *)
00380 LABEL_NSWEEPS,
00381                             input->variable[i].nsweeps);
00382     else if (input->algorithm == ALGORITHM_GENETIC)
00383         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00384                             input->variable[i].nbits);
00385     if (input->nsteps)
00386         xml_node_set_float (child, (const xmlChar *)
00387 LABEL_STEP,
00388                             input->variable[i].step);
00389 }

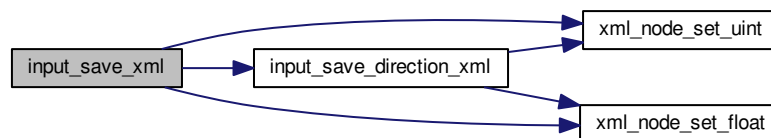
```

```

00382
00383 // Saving the error norm
00384 switch (input->norm)
00385 {
00386     case ERROR_NORM_MAXIMUM:
00387         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00388                     (const xmlChar *) LABEL_MAXIMUM);
00389         break;
00390     case ERROR_NORM_P:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                     (const xmlChar *) LABEL_P);
00393         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00394                             input->p);
00395         break;
00396     case ERROR_NORM_TAXICAB:
00397         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00398                     (const xmlChar *) LABEL_TAXICAB);
00399 }
00400 #if DEBUG_INTERFACE
00401 fprintf (stderr, "input_save: end\n");
00402 #endif
00403 }

```

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

```

00726 {
00727     unsigned int i;
00728     #if DEBUG_INTERFACE
00729         fprintf (stderr, "window_get_algorithm: start\n");
00730     #endif
00731     i = gtk_array_get_active (window->button_algorithm,
00732                             NALEGORITHMS);
00733     #if DEBUG_INTERFACE
00734         fprintf (stderr, "window_get_algorithm: %u\n", i);
00735         fprintf (stderr, "window_get_algorithm: end\n");
00736     #endif
00737     return i;
00738 }

```

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

```
00746 {  
00747     unsigned int i;  
00748     #if DEBUG_INTERFACE  
00749     fprintf (stderr, "window_get_direction: start\n");  
00750     #endif  
00751     i = gtk_array_get_active (window->button_direction,  
00752                             NDIRECTIONS);  
00752     #if DEBUG_INTERFACE  
00753     fprintf (stderr, "window_get_direction: %u\n", i);  
00754     fprintf (stderr, "window_get_direction: end\n");  
00755     #endif  
00756     return i;  
00757 }
```

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

```
00766 {
00767     unsigned int i;
00768     #if DEBUG_INTERFACE
00769     fprintf (stderr, "window_get_norm: start\n");
00770     #endif
00771     i = gtk_array_get_active (window->button_norm,
00772                             NNORMS);
00773     #if DEBUG_INTERFACE
00774     fprintf (stderr, "window_get_norm: %u\n", i);
00775     fprintf (stderr, "window_get_norm: end\n");
00776     #endif
00777     return i;
00778 }
```

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 2081 of file [interface.c](#).

```
02082 {
```

```

02083 unsigned int i;
02084 char *buffer, *buffer2, buffer3[64];
02085 char *label_algorithm[NALGORITHMS] = {
02086     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02087 };
02088 char *tip_algorithm[NALGORITHMS] = {
02089     _("Monte-Carlo brute force algorithm"),
02090     _("Sweep brute force algorithm"),
02091     _("Genetic algorithm")
02092 };
02093 char *label_direction[NDIRECTIONS] = {
02094     _("_Coordinates descent"), _("_Random")
02095 };
02096 char *tip_direction[NDIRECTIONS] = {
02097     _("Coordinates direction estimate method"),
02098     _("Random direction estimate method")
02099 };
02100 char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02101 char *tip_norm[NNORMS] = {
02102     _("Euclidean error norm (L2)"),
02103     _("Maximum error norm (L)"),
02104     _("P error norm (Lp)"),
02105     _("Taxicab error norm (L1)")
02106 };
02107
02108 #if DEBUG_INTERFACE
02109     fprintf (stderr, "window_new: start\n");
02110 #endif
02111
02112 // Creating the window
02113 window->window = main_window
02114     = (GtkWindow *) gtk_application_window_new (application);
02115
02116 // Finish when closing the window
02117 g_signal_connect_swapped (window->window, "delete-event",
02118     G_CALLBACK (g_application_quit),
02119     G_APPLICATION (application));
02120
02121 // Setting the window title
02122 gtk_window_set_title (window->window, "MPCOTool");
02123
02124 // Creating the open button
02125 window->button_open = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("document-open",
02127     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02128 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02129
02130 // Creating the save button
02131 window->button_save = (GtkToolButton *) gtk_tool_button_new
02132     (gtk_image_new_from_icon_name ("document-save",
02133     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02134 g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
02135     NULL);
02136
02137 // Creating the run button
02138 window->button_run = (GtkToolButton *) gtk_tool_button_new
02139     (gtk_image_new_from_icon_name ("system-run",
02140     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02141 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02142
02143 // Creating the options button
02144 window->button_options = (GtkToolButton *) gtk_tool_button_new
02145     (gtk_image_new_from_icon_name ("preferences-system",
02146     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02147 g_signal_connect (window->button_options, "clicked",
options_new, NULL);
02148
02149 // Creating the help button
02150 window->button_help = (GtkToolButton *) gtk_tool_button_new
02151     (gtk_image_new_from_icon_name ("help-browser",
02152     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02153 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02154
02155 // Creating the about button
02156 window->button_about = (GtkToolButton *) gtk_tool_button_new
02157     (gtk_image_new_from_icon_name ("help-about",
02158     GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02159 g_signal_connect (window->button_about, "clicked",
window_about, NULL);
02160
02161 // Creating the exit button
02162 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02163     (gtk_image_new_from_icon_name ("application-exit",
02164     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02165 g_signal_connect_swapped (window->button_exit, "clicked",
G_CALLBACK (g_application_quit),
02166

```

```

02167             G_APPLICATION (application));
02168
02169 // Creating the buttons bar
02170 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02171 gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_open), 0);
02173 gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_save), 1);
02175 gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_run), 2);
02177 gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_options), 3);
02179 gtk_toolbar_insert
02180     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_help), 4);
02181 gtk_toolbar_insert
02182     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_about), 5);
02183 gtk_toolbar_insert
02184     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_exit), 6);
02185 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02186
02187 // Creating the simulator program label and entry
02188 window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02189 window->button_simulator = (GtkFileChooserButton *)
02190     gtk_file_chooser_button_new (_("Simulator program"),
GTK_FILE_CHOOSER_ACTION_OPEN);
02191 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02192     _("Simulator program executable file"));
02193 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02194
02195 // Creating the evaluator program label and entry
02196 window->check_evaluator = (GtkCheckButton *)
02197     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02198 g_signal_connect (window->check_evaluator, "toggled",
02199     window_update, NULL);
02200 window->button_evaluator = (GtkFileChooserButton *)
02201     gtk_file_chooser_button_new (_("Evaluator program"),
GTK_FILE_CHOOSER_ACTION_OPEN);
02202 gtk_widget_set_tooltip_text
02203     (GTK_WIDGET (window->button_evaluator),
02204     _("Optional evaluator program executable file"));
02205
02206 // Creating the results files labels and entries
02207 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02208 window->entry_result = (GtkEntry *) gtk_entry_new ();
02209 gtk_widget_set_tooltip_text
02210     (GTK_WIDGET (window->entry_result), _("Best results file"));
02211 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02212 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02213 gtk_widget_set_tooltip_text
02214     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02215
02216 // Creating the files grid and attaching widgets
02217 window->grid_files = (GtkGrid *) gtk_grid_new ();
02218 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02219     0, 0, 1, 1);
02220 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02221     1, 0, 1, 1);
02222 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02223     0, 1, 1, 1);
02224 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02225     1, 1, 1, 1);
02226 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02227     0, 2, 1, 1);
02228 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02229     1, 2, 1, 1);
02230 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02231     0, 3, 1, 1);
02232 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02233     1, 3, 1, 1);
02234
02235 // Creating the algorithm properties
02236 window->label_simulations = (GtkLabel *) gtk_label_new

```

```

02238     _("Simulations number"));
02239 window->spin_simulations
02240     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02241 gtk_widget_set_tooltip_text
02242     (GTK_WIDGET (window->spin_simulations),
02243      _("Number of simulations to perform for each iteration"));
02244 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02245 window->label_iterations = (GtkLabel *)
02246     gtk_label_new (_("Iterations number"));
02247 window->spin_iterations
02248     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02249 gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02251 g_signal_connect
02252     (window->spin_iterations, "value-changed",
window_update, NULL);
02253 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02254 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02255 window->spin_tolerance =
02256     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02257 gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_tolerance),
02259      _("Tolerance to set the variable interval on the next iteration"));
02260 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02261 window->spin_bests
02262     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02263 gtk_widget_set_tooltip_text
02264     (GTK_WIDGET (window->spin_bests),
02265      _("Number of best simulations used to set the variable interval "
02266        "on the next iteration"));
02267 window->label_population
02268     = (GtkLabel *) gtk_label_new (_("Population number"));
02269 window->spin_population
02270     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02271 gtk_widget_set_tooltip_text
02272     (GTK_WIDGET (window->spin_population),
02273      _("Number of population for the genetic algorithm"));
02274 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02275 window->label_generations
02276     = (GtkLabel *) gtk_label_new (_("Generations number"));
02277 window->spin_generations
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02279 gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_generations),
02281      _("Number of generations for the genetic algorithm"));
02282 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02283 window->spin_mutation
02284     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02285 gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_mutation),
02287      _("Ratio of mutation for the genetic algorithm"));
02288 window->label_reproduction
02289     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02290 window->spin_reproduction
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292 gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_reproduction),
02294      _("Ratio of reproduction for the genetic algorithm"));
02295 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02296 window->spin_adaptation
02297     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02298 gtk_widget_set_tooltip_text
02299     (GTK_WIDGET (window->spin_adaptation),
02300      _("Ratio of adaptation for the genetic algorithm"));
02301 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02302 window->spin_threshold = (GtkSpinButton *)
02303     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02304                                     precision[DEFAULT_PRECISION]);
02305 gtk_widget_set_tooltip_text
02306     (GTK_WIDGET (window->spin_threshold),
02307      _("Threshold in the objective function to finish the simulations"));
02308 window->scrolled_threshold =
02309     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02310 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02311                   GTK_WIDGET (window->spin_threshold));
02312 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02313 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02314 //                          GTK_ALIGN_FILL);
02315
02316 // Creating the direction search method properties
02317 window->check_direction = (GtkCheckButton *)
02318     gtk_check_button_new_with_mnemonic (_("Direction search method"));
02319 g_signal_connect (window->check_direction, "clicked",
window_update, NULL);
02320 window->grid_direction = (GtkGrid *) gtk_grid_new ();
02321 window->button_direction[0] = (GtkRadioButton *)
02322     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);

```

```

02323     gtk_grid_attach (window->grid_direction,
02324                     GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02325     g_signal_connect (window->button_direction[0], "clicked",
window_update,
02326                     NULL);
02327     for (i = 0; ++i < NDIRECTIONS;)
02328     {
02329         window->button_direction[i] = (GtkRadioButton *)
02330             gtk_radio_button_new_with_mnemonic
02331             (gtk_radio_button_get_group (window->button_direction[0]),
02332              label_direction[i]);
02333         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02334                                     tip_direction[i]);
02335         gtk_grid_attach (window->grid_direction,
02336                         GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02337         g_signal_connect (window->button_direction[i], "clicked",
02338                         window_update, NULL);
02339     }
02340     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02341     window->spin_steps = (GtkSpinButton *)
02342         gtk_spin_button_new_with_range (1., 1.e12, 1.);
02343     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02344     window->label_estimates
02345         = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02346     window->spin_estimates = (GtkSpinButton *)
02347         gtk_spin_button_new_with_range (1., 1.e3, 1.);
02348     window->label_relaxation
02349         = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02350     window->spin_relaxation = (GtkSpinButton *)
02351         gtk_spin_button_new_with_range (0., 2., 0.001);
02352     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
window->label_steps),
02353                     0, NDIRECTIONS, 1, 1);
02354     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
window->spin_steps),
02355                     1, NDIRECTIONS, 1, 1);
02356     gtk_grid_attach (window->grid_direction,
02357                     GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02358                     1, 1);
02359     gtk_grid_attach (window->grid_direction,
02360                     GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02361                     1);
02362     gtk_grid_attach (window->grid_direction,
02363                     GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02364                     1, 1);
02365     gtk_grid_attach (window->grid_direction,
02366                     GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02367                     1, 1);
02368
02369     // Creating the array of algorithms
02370     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02371     window->button_algorithm[0] = (GtkRadioButton *)
02372         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02373     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02374                                 tip_algorithm[0]);
02375     gtk_grid_attach (window->grid_algorithm,
02376                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02377     g_signal_connect (window->button_algorithm[0], "clicked",
02378                     window_set_algorithm, NULL);
02379     for (i = 0; ++i < NALGORITHMS;)
02380     {
02381         window->button_algorithm[i] = (GtkRadioButton *)
02382             gtk_radio_button_new_with_mnemonic
02383             (gtk_radio_button_get_group (window->button_algorithm[0]),
02384              label_algorithm[i]);
02385         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02386                                     tip_algorithm[i]);
02387         gtk_grid_attach (window->grid_algorithm,
02388                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02389         g_signal_connect (window->button_algorithm[i], "clicked",
02390                         window_set_algorithm, NULL);
02391     }
02392     gtk_grid_attach (window->grid_algorithm,
02393                     GTK_WIDGET (window->label_simulations), 0,
02394                     NALGORITHMS, 1, 1);
02395     gtk_grid_attach (window->grid_algorithm,
02396                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02397     gtk_grid_attach (window->grid_algorithm,
02398                     GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02399                     1, 1);
02400     gtk_grid_attach (window->grid_algorithm,
02401                     GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02402                     1, 1);
02403     gtk_grid_attach (window->grid_algorithm,
02404                     GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02405                     1, 1);
02406     gtk_grid_attach (window->grid_algorithm,

```



```

02407         GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02408         1);
02409     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02410         window->label_bests),
02411         0, NALGORITHMS + 3, 1, 1);
02412     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02413         window->spin_bests), 1,
02414         NALGORITHMS + 3, 1, 1);
02415     gtk_grid_attach (window->grid_algorithm,
02416         GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02417         1, 1);
02418     gtk_grid_attach (window->grid_algorithm,
02419         GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02420         1, 1);
02421     gtk_grid_attach (window->grid_algorithm,
02422         GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02423         1, 1);
02424     gtk_grid_attach (window->grid_algorithm,
02425         GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02426         1, 1);
02427     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02428         window->spin_mutation),
02429         1, NALGORITHMS + 6, 1, 1);
02430     gtk_grid_attach (window->grid_algorithm,
02431         GTK_WIDGET (window->label_reproduction), 0,
02432         NALGORITHMS + 7, 1, 1);
02433     gtk_grid_attach (window->grid_algorithm,
02434         GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02435         1, 1);
02436     gtk_grid_attach (window->grid_algorithm,
02437         GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02438         1, 1);
02439     gtk_grid_attach (window->grid_algorithm,
02440         GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02441         1, 1);
02442     gtk_grid_attach (window->grid_algorithm,
02443         GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02444         2, 1);
02445     gtk_grid_attach (window->grid_algorithm,
02446         GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02447         2, 1);
02448     gtk_grid_attach (window->grid_algorithm,
02449         GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02450         1, 1);
02451     gtk_grid_attach (window->grid_algorithm,
02452         GTK_WIDGET (window->scrolled_threshold), 1,
02453         NALGORITHMS + 11, 1, 1);
02454     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02455     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02456         GTK_WIDGET (window->grid_algorithm));
02457
02458     // Creating the variable widgets
02459     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02460     gtk_widget_set_tooltip_text
02461         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02462     window->id_variable = g_signal_connect
02463         (window->combo_variable, "changed", window_set_variable, NULL);
02464     window->button_add_variable
02465         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02466             GTK_ICON_SIZE_BUTTON);
02467     g_signal_connect
02468         (window->button_add_variable, "clicked",
02469         window_add_variable, NULL);
02470     gtk_widget_set_tooltip_text
02471         (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02472     window->button_remove_variable
02473         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02474             GTK_ICON_SIZE_BUTTON);
02475     g_signal_connect
02476         (window->button_remove_variable, "clicked",
02477         window_remove_variable, NULL);
02478     gtk_widget_set_tooltip_text
02479         (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02480     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02481     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02482     gtk_widget_set_tooltip_text
02483         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02484     gtk_widget_set_hexexpand (GTK_WIDGET (window->entry_variable), TRUE);
02485     window->id_variable_label = g_signal_connect
02486         (window->entry_variable, "changed",
02487         window_label_variable, NULL);
02488     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02489     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02490         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);

```

```

02488 gtk_widget_set_tooltip_text
02489     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02490 window->scrolled_min
02491     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02492 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02493     GTK_WIDGET (window->spin_min));
02494 g_signal_connect (window->spin_min, "value-changed",
02495     window_rangemin_variable, NULL);
02496 window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02497 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02498     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02499 gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02501 window->scrolled_max
02502     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02503 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02504     GTK_WIDGET (window->spin_max));
02505 g_signal_connect (window->spin_max, "value-changed",
02506     window_rangemax_variable, NULL);
02507 window->check_minabs = (GtkCheckButton *)
02508     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02509 g_signal_connect (window->check_minabs, "toggled",
02510     window_update, NULL);
02511 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02512     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02513 gtk_widget_set_tooltip_text
02514     (GTK_WIDGET (window->spin_minabs),
02515     _("Minimum allowed value of the variable"));
02516 window->scrolled_minabs
02517     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02518 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02519     GTK_WIDGET (window->spin_minabs));
02520 g_signal_connect (window->spin_minabs, "value-changed",
02521     window_rangeminabs_variable, NULL);
02522 window->check_maxabs = (GtkCheckButton *)
02523     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02524 g_signal_connect (window->check_maxabs, "toggled",
02525     window_update, NULL);
02526 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02527     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02528 gtk_widget_set_tooltip_text
02529     (GTK_WIDGET (window->spin_maxabs),
02530     _("Maximum allowed value of the variable"));
02531 window->scrolled_maxabs
02532     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02533 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02534     GTK_WIDGET (window->spin_maxabs));
02535 g_signal_connect (window->spin_maxabs, "value-changed",
02536     window_rangemaxabs_variable, NULL);
02537 window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02538 window->spin_precision = (GtkSpinButton *)
02539     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02540 gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->spin_precision),
02542     _("Number of precision floating point digits\n"
02543     "0 is for integer numbers"));
02544 g_signal_connect (window->spin_precision, "value-changed",
02545     window_precision_variable, NULL);
02546 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02547 window->spin_sweeps =
02548     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02549 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02550     _("Number of steps sweeping the variable"));
02551 g_signal_connect (window->spin_sweeps, "value-changed",
02552     window_update_variable, NULL);
02553 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02554 window->spin_bits
02555     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02556 gtk_widget_set_tooltip_text
02557     (GTK_WIDGET (window->spin_bits),
02558     _("Number of bits to encode the variable"));
02559 g_signal_connect
02560     (window->spin_bits, "value-changed", window_update_variable, NULL);
02561 ;
02562 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02563 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02564     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02565 gtk_widget_set_tooltip_text
02566     (GTK_WIDGET (window->spin_step),
02567     _("Initial step size for the direction search method"));
02568 window->scrolled_step
02569     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02570 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02571     GTK_WIDGET (window->spin_step));
02572 g_signal_connect
02573     (window->spin_step, "value-changed", window_step_variable, NULL);
02574 window->grid_variable = (GtkGrid *) gtk_grid_new ();

```

```

02572 gtk_grid_attach (window->grid_variable,
02573                 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02574 gtk_grid_attach (window->grid_variable,
02575                 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02576 gtk_grid_attach (window->grid_variable,
02577                 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02578 gtk_grid_attach (window->grid_variable,
02579                 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02580 gtk_grid_attach (window->grid_variable,
02581                 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02582 gtk_grid_attach (window->grid_variable,
02583                 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02584 gtk_grid_attach (window->grid_variable,
02585                 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02586 gtk_grid_attach (window->grid_variable,
02587                 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02588 gtk_grid_attach (window->grid_variable,
02589                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02590 gtk_grid_attach (window->grid_variable,
02591                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02592 gtk_grid_attach (window->grid_variable,
02593                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02594 gtk_grid_attach (window->grid_variable,
02595                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02596 gtk_grid_attach (window->grid_variable,
02597                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02598 gtk_grid_attach (window->grid_variable,
02599                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02600 gtk_grid_attach (window->grid_variable,
02601                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02602 gtk_grid_attach (window->grid_variable,
02603                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02604 gtk_grid_attach (window->grid_variable,
02605                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02606 gtk_grid_attach (window->grid_variable,
02607                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02608 gtk_grid_attach (window->grid_variable,
02609                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02610 gtk_grid_attach (window->grid_variable,
02611                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02614 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02615 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02616                   GTK_WIDGET (window->grid_variable));
02617
02618 // Creating the experiment widgets
02619 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02620 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02621                             _("Experiment selector"));
02622 window->id_experiment = g_signal_connect
02623 (window->combo_experiment, "changed",
02624  window_set_experiment, NULL);
02625 window->button_add_experiment
02626 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02627                                                GTK_ICON_SIZE_BUTTON);
02628 g_signal_connect
02629 (window->button_add_experiment, "clicked",
02630  window_add_experiment, NULL);
02631 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02632                             _("Add experiment"));
02633 window->button_remove_experiment
02634 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02635                                                GTK_ICON_SIZE_BUTTON);
02636 g_signal_connect (window->button_remove_experiment, "clicked",
02637                  window_remove_experiment, NULL);
02638 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
02639 button_remove_experiment),
02640                             _("Remove experiment"));
02641 window->label_experiment
02642 = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02643 window->button_experiment = (GtkFileChooserButton *)
02644   gtk_file_chooser_button_new (_("Experimental data file"),
02645                               GTK_FILE_CHOOSER_ACTION_OPEN);
02646 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02647                             _("Experimental data file"));
02648 window->id_experiment_name
02649 = g_signal_connect (window->button_experiment, "selection-changed",
02650                    window_name_experiment, NULL);
02651 gtk_widget_set_hexand (GTK_WIDGET (window->button_experiment), TRUE);
02652 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02653 window->spin_weight
02654 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02655 gtk_widget_set_tooltip_text
02656 (GTK_WIDGET (window->spin_weight),
02657  _("Weight factor to build the objective function"));
02658 g_signal_connect

```

```

02656     (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
02657 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02658 gtk_grid_attach (window->grid_experiment,
02659     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02660 gtk_grid_attach (window->grid_experiment,
02661     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02662 gtk_grid_attach (window->grid_experiment,
02663     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
;
02664 gtk_grid_attach (window->grid_experiment,
02665     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02666 gtk_grid_attach (window->grid_experiment,
02667     GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02668 gtk_grid_attach (window->grid_experiment,
02669     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02670 gtk_grid_attach (window->grid_experiment,
02671     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02672 for (i = 0; i < MAX_NINPUTS; ++i)
02673 {
02674     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02675     window->check_template[i] = (GtkCheckButton *)
02676     gtk_check_button_new_with_label (buffer3);
02677     window->id_template[i]
02678     = g_signal_connect (window->check_template[i], "toggled",
02679     window_inputs_experiment, NULL);
02680     gtk_grid_attach (window->grid_experiment,
02681     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02682     window->button_template[i] =
02683     (GtkFileChooserButton *)
02684     gtk_file_chooser_button_new (_("Input template"),
02685     GTK_FILE_CHOOSER_ACTION_OPEN);
02686     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02687     _("Experimental input template file"));
02688     window->id_input[i] =
02689     g_signal_connect_swapped (window->button_template[i],
02690     "selection-changed",
02691     (GCallback) window_template_experiment,
02692     (void *) (size_t) i);
02693     gtk_grid_attach (window->grid_experiment,
02694     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02695 }
02696 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02697 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02698     GTK_WIDGET (window->grid_experiment));
02699
02700 // Creating the error norm widgets
02701 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02702 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02703 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02704     GTK_WIDGET (window->grid_norm));
02705 window->button_norm[0] = (GtkRadioButton *)
02706     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02707 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02708     tip_norm[0]);
02709 gtk_grid_attach (window->grid_norm,
02710     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02711 g_signal_connect (window->button_norm[0], "clicked",
window_update, NULL);
02712 for (i = 0; ++i < NNORMS;)
02713 {
02714     window->button_norm[i] = (GtkRadioButton *)
02715     gtk_radio_button_new_with_mnemonic
02716     (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02717     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02718     tip_norm[i]);
02719     gtk_grid_attach (window->grid_norm,
02720     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02721     g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
02722 }
02723 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02724 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
label_p), 1, 1, 1, 1);
02725 window->spin_p =
02726     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02727     G_MAXDOUBLE, 0.01);
02728 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02729     _("P parameter for the P error norm"));
02730 window->scrolled_p =
02731     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02732 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02733     GTK_WIDGET (window->spin_p));
02734 gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02735 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02736 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
scrolled_p),

```

```

02737         1, 2, 1, 2);
02738
02739 // Creating the grid and attaching the widgets to the grid
02740 window->grid = (GtkGrid *) gtk_grid_new ();
02741 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02742 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02743 gtk_grid_attach (window->grid,
02744                 GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02745 gtk_grid_attach (window->grid,
02746                 GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02747 gtk_grid_attach (window->grid,
02748                 GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02749 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02750 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
02751 window->grid));
02752
02752 // Setting the window logo
02753 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02754 gtk_window_set_icon (window->window, window->logo);
02755
02756 // Showing the window
02757 gtk_widget_show_all (GTK_WIDGET (window->window));
02758
02759 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02760 #if GTK_MINOR_VERSION >= 16
02761 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02762 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02763 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02764 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02765 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02766 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02767 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02768 #endif
02769
02770 // Reading initial example
02771 input_new ();
02772 buffer2 = g_get_current_dir ();
02773 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02774 g_free (buffer2);
02775 window_read (buffer);
02776 g_free (buffer);
02777
02778 #if DEBUG_INTERFACE
02779 fprintf (stderr, "window_new: start\n");
02780 #endif
02781 }

```

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 1877 of file [interface.c](#).

```

01878 {
01879     unsigned int i;
01880     char *buffer;

```

```

01881 #if DEBUG_INTERFACE
01882     fprintf (stderr, "window_read: start\n");
01883 #endif
01884
01885     // Reading new input file
01886     input_free ();
01887     input->result = input->variables = NULL;
01888     if (!input_open (filename))
01889     {
01890 #if DEBUG_INTERFACE
01891         fprintf (stderr, "window_read: end\n");
01892 #endif
01893         return 0;
01894     }
01895
01896     // Setting GTK+ widgets data
01897     gtk_entry_set_text (window->entry_result, input->result);
01898     gtk_entry_set_text (window->entry_variables, input->
variables);
01899     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01900     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01901     g_free (buffer);
01902     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01903     if (input->evaluator)
01904     {
01905         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01906         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01907         g_free (buffer);
01908     }
01909     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01910     switch (input->algorithm)
01911     {
01912     case ALGORITHM_MONTE_CARLO:
01913         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01914     case ALGORITHM_SWEEP:
01915         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01916         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01917         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01918         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_direction),
input->nsteps);
01919         if (input->nsteps)
01920         {
01921             gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01922             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01923             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01924             switch (input->direction)
01925             {
01926             case DIRECTION_METHOD_RANDOM:
01927                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01928             }
01929         }
01930         break;
01931     default:
01932         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
01933         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
01934         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01935         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
01936         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
01937     }
01938     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01939     gtk_spin_button_set_value (window->spin_p, input->p);
01940     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01941     g_signal_handler_block (window->combo_experiment, window->

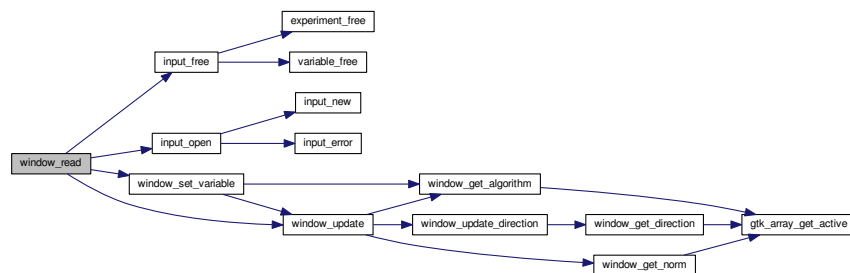
```

```

    id_experiment);
01959 g_signal_handler_block (window->button_experiment,
01960                          window->id_experiment_name);
01961 gtk_combo_box_text_remove_all (window->combo_experiment);
01962 for (i = 0; i < input->nexperiments; ++i)
01963     gtk_combo_box_text_append_text (window->combo_experiment,
01964                                   input->experiment[i].name);
01965 g_signal_handler_unblock
01966     (window->button_experiment, window->
id_experiment_name);
01967 g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01968 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01969 g_signal_handler_block (window->combo_variable, window->
id_variable);
01970 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01971 gtk_combo_box_text_remove_all (window->combo_variable);
01972 for (i = 0; i < input->nvariables; ++i)
01973     gtk_combo_box_text_append_text (window->combo_variable,
01974                                   input->variable[i].name);
01975 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01976 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01977 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01978 window_set_variable ();
01979 window_update ();
01980
01981 #if DEBUG_INTERFACE
01982 fprintf (stderr, "window_read: end\n");
01983 #endif
01984 return 1;
01985 }

```

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

```

00819 {
00820     GtkFileChooserDialog *dlg;
00821     GtkFileFilter *filter1, *filter2;
00822     char *buffer;
00823
00824     #if DEBUG_INTERFACE
00825         fprintf (stderr, "window_save: start\n");
00826     #endif
00827
00828     // Opening the saving dialog
00829     dlg = (GtkFileChooserDialog *)
00830         gtk_file_chooser_dialog_new (_("Save file"),
00831                                     window->window,
00832                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00833                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
00834                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00835     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00836     buffer = g_build_filename (input->directory, input->name, NULL);
00837     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00838     g_free (buffer);
00839
00840     // Adding XML filter
00841     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00842     gtk_file_filter_set_name (filter1, "XML");
00843     gtk_file_filter_add_pattern (filter1, "*.xml");
00844     gtk_file_filter_add_pattern (filter1, "*.XML");
00845     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00846
00847     // Adding JSON filter
00848     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00849     gtk_file_filter_set_name (filter2, "JSON");
00850     gtk_file_filter_add_pattern (filter2, "*.json");
00851     gtk_file_filter_add_pattern (filter2, "*.JSON");
00852     gtk_file_filter_add_pattern (filter2, "*.js");
00853     gtk_file_filter_add_pattern (filter2, "*.JS");
00854     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00855
00856     if (input->type == INPUT_TYPE_XML)
00857         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00858     else
00859         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861     // If OK response then saving
00862     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00863     {
00864         // Setting input file type
00865         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866         buffer = (char *) gtk_file_filter_get_name (filter1);
00867         if (!strcmp (buffer, "XML"))
00868             input->type = INPUT_TYPE_XML;
00869         else
00870             input->type = INPUT_TYPE_JSON;
00871
00872         // Adding properties to the root XML node
00873         input->simulator = gtk_file_chooser_get_filename
00874             (GTK_FILE_CHOOSER (window->button_simulator));
00875         if (gtk_toggle_button_get_active
00876             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00877             input->evaluator = gtk_file_chooser_get_filename
00878                 (GTK_FILE_CHOOSER (window->button_evaluator));
00879         else
00880             input->evaluator = NULL;
00881         if (input->type == INPUT_TYPE_XML)
00882         {
00883             input->result
00884                 = (char *) xmlStrdup ((const xmlChar *)
00885                                         gtk_entry_get_text (window->entry_result));
00886             input->variables
00887                 = (char *) xmlStrdup ((const xmlChar *)
00888                                         gtk_entry_get_text (window->
00889 entry_variables));
00890         }
00891         else
00892         {
00893             input->result = g_strdup (gtk_entry_get_text (window->
00894 entry_result));
00895             input->variables =
00896                 g_strdup (gtk_entry_get_text (window->entry_variables));
00897         }
00898         // Setting the algorithm
00899         switch (window_get_algorithm ())
00900         {
00901             case ALGORITHM_MONTE_CARLO:
00902                 input->algorithm = ALGORITHM_MONTE_CARLO;
00903                 input->nsimulations
00904                     = gtk_spin_button_get_value_as_int (window->spin_simulations);

```



```

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

```

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 1521 of file [interface.c](#).

```

01522 {
01523     unsigned int i, j;
01524     char *buffer;
01525     GFile *file1, *file2;
01526     #if DEBUG_INTERFACE
01527     fprintf (stderr, "window_template_experiment: start\n");
01528     #endif
01529     i = (size_t) data;
01530     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01531     file1
01532     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01533     file2 = g_file_new_for_path (input->directory);
01534     buffer = g_file_get_relative_path (file2, file1);
01535     if (input->type == INPUT_TYPE_XML)
01536         input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01537     else
01538         input->experiment[j].stencil[i] = g_strdup (buffer);
01539     g_free (buffer);
01540     g_object_unref (file2);
01541     g_object_unref (file1);
01542     #if DEBUG_INTERFACE
01543     fprintf (stderr, "window_template_experiment: end\n");
01544     #endif
01545 }

```

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 2012-2017, 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"

```

```

00062 #include "input.h"
00063 #include "optimize.h"
00064 #include "interface.h"
00065
00066 #define DEBUG_INTERFACE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define INPUT_FILE "test-ga-win.xml"
00074 #else
00075 #define INPUT_FILE "test-ga.xml"
00076 #endif
00077
00078 const char *logo[] = {
00079     "32 32 3 1",
00080     "    c None",
00081     ".    c #0000FF",
00082     "+    c #FF0000",
00083     "                                ",
00084     "                                ",
00085     "                                ",
00086     "    .    .    .    .    ",
00087     "    .    .    .    .    ",
00088     "    .    .    .    .    ",
00089     "    .    .    .    .    ",
00090     "    .    .    +++    .    ",
00091     "    .    .    +++++    .    ",
00092     "    .    .    +++++    .    ",
00093     "    .    .    +++++    .    ",
00094     "    +++    .    +++    +++    ",
00095     "    +++++    .    +++++    ",
00096     "    +++++    .    +++++    ",
00097     "    +++++    .    +++++    ",
00098     "    +++    .    +++    ",
00099     "    .    .    .    .    ",
00100     "    .    +++    .    .    ",
00101     "    .    +++++    .    .    ",
00102     "    .    +++++    .    .    ",
00103     "    .    +++++    .    .    ",
00104     "    .    +++    .    .    ",
00105     "    .    .    .    .    ",
00106     "    .    .    .    .    ",
00107     "    .    .    .    .    ",
00108     "    .    .    .    .    ",
00109     "    .    .    .    .    ",
00110     "    .    .    .    .    ",
00111     "    .    .    .    .    ",
00112     "                                ",
00113     "                                ",
00114     "                                ",
00115 };
00116
00117 /*
00118 const char * logo[] = {
00119     "32 32 3 1",
00120     "    c #FFFFFFFF",
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,
input->nsteps);
00179         if (input->relaxation != DEFAULT_RELAXATION)
00180             xml_node_set_float (node, (const xmlChar *)
LABEL_RELAXATION,
input->relaxation);
00181         switch (input->direction)
00182         {
00183             case DIRECTION_METHOD_COORDINATES:
00184                 xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
(const xmlChar *) LABEL_COORDINATES);
00185                 break;
00186             default:
00187                 xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
(const xmlChar *) LABEL_RANDOM);
00188                 xml_node_set_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
input->nestimates);
00189         }
00190     }
00191     #if DEBUG_INTERFACE
00192         fprintf (stderr, "input_save_direction_xml: end\n");
00193     #endif
00194 }
00195
00206 void
00207 input_save_direction_json (JsonNode * node)
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,
input->nsteps);
00217         if (input->relaxation != DEFAULT_RELAXATION)
00218             json_object_set_float (object, LABEL_RELAXATION,
input->relaxation);
00219         switch (input->direction)
00220         {
00221             case DIRECTION_METHOD_COORDINATES:
00222                 json_object_set_string_member (object, LABEL_DIRECTION,
LABEL_COORDINATES);
00223                 break;
00224             default:
00225                 json_object_set_string_member (object, LABEL_DIRECTION,
LABEL_RANDOM);
00226                 json_object_set_uint (object, LABEL_NESTIMATES,
input->nestimates);
00227         }
00228     }
00229     #if DEBUG_INTERFACE
00230         fprintf (stderr, "input_save_direction_json: end\n");
00231     #endif
00232 }
00233
00241 void
00242 input_save_xml (xmlDoc * doc)
00243 {
00244     unsigned int i, j;
00245     char *buffer;
00246     xmlNode *node, *child;
00247     GFile *file, *file2;
00248     #if DEBUG_INTERFACE
00249         fprintf (stderr, "input_save_xml: start\n");
00250     #endif
00251 }
00252

```

```

00253 // Setting root XML node
00254 node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00255 xmlDocSetRootElement (doc, node);
00256
00257 // Adding properties to the root XML node
00258 if (xmlStrcmp
00259     ((const xmlChar *) input->result, (const xmlChar *) result_name))
00260     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00261                 (xmlChar *) input->result);
00262 if (xmlStrcmp
00263     ((const xmlChar *) input->variables, (const xmlChar *)
variables_name))
00264     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00265                 (xmlChar *) input->variables);
00266 file = g_file_new_for_path (input->directory);
00267 file2 = g_file_new_for_path (input->simulator);
00268 buffer = g_file_get_relative_path (file, file2);
00269 g_object_unref (file2);
00270 xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00271 g_free (buffer);
00272 if (input->evaluator)
00273 {
00274     file2 = g_file_new_for_path (input->evaluator);
00275     buffer = g_file_get_relative_path (file, file2);
00276     g_object_unref (file2);
00277     if (xmlStrlen ((xmlChar *) buffer))
00278         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00279                     (xmlChar *) buffer);
00280     g_free (buffer);
00281 }
00282 if (input->seed != DEFAULT_RANDOM_SEED)
00283     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);
00284
00285 // Setting the algorithm
00286 buffer = (char *) g_slice_alloc (64);
00287 switch (input->algorithm)
00288 {
00289     case ALGORITHM_MONTE_CARLO:
00290         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00291                     (const xmlChar *) LABEL_MONTE_CARLO);
00292         snprintf (buffer, 64, "%u", input->nsimulations);
00293         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00294                     (xmlChar *) buffer);
00295         snprintf (buffer, 64, "%u", input->niterations);
00296         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00297                     (xmlChar *) buffer);
00298         snprintf (buffer, 64, "%.3lg", input->tolerance);
00299         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%u", input->nbest);
00301         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00302         input_save_direction_xml (node);
00303         break;
00304     case ALGORITHM_SWEEP:
00305         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00306                     (const xmlChar *) LABEL_SWEEP);
00307         snprintf (buffer, 64, "%u", input->niterations);
00308         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00309                     (xmlChar *) buffer);
00310         snprintf (buffer, 64, "%.3lg", input->tolerance);
00311         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00312         snprintf (buffer, 64, "%u", input->nbest);
00313         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00314         input_save_direction_xml (node);
00315         break;
00316     default:
00317         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00318                     (const xmlChar *) LABEL_GENETIC);
00319         snprintf (buffer, 64, "%u", input->nsimulations);
00320         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00321                     (xmlChar *) buffer);
00322         snprintf (buffer, 64, "%u", input->niterations);
00323         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00324                     (xmlChar *) buffer);
00325         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00326         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00328         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00329                     (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00332         break;
00333 }
00334 g_slice_free1 (64, buffer);
00335 if (input->threshold != 0.)
00336     xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,

```

```

00337         input->threshold);
00338
00339 // Setting the experimental data
00340 for (i = 0; i < input->nexperiments; ++i)
00341 {
00342     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00343     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00344                 (xmlChar *) input->experiment[i].name);
00345     if (input->experiment[i].weight != 1.)
00346         xml_node_set_float (child, (const xmlChar *)
00347                             LABEL_WEIGHT,
00348                             input->experiment[i].weight);
00349     for (j = 0; j < input->experiment->ninputs; ++j)
00350         xmlSetProp (child, (const xmlChar *) stencil[j],
00351                     (xmlChar *) input->experiment[i].stencil[j]);
00352 }
00353 // Setting the variables data
00354 for (i = 0; i < input->nvariables; ++i)
00355 {
00356     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00357     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00358                 (xmlChar *) input->variable[i].name);
00359     xml_node_set_float (child, (const xmlChar *)
00360                         LABEL_MINIMUM,
00361                         input->variable[i].rangemin);
00362     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00363         xml_node_set_float (child, (const xmlChar *)
00364                             LABEL_ABSOLUTE_MINIMUM,
00365                             input->variable[i].rangeminabs);
00366     xml_node_set_float (child, (const xmlChar *)
00367                         LABEL_MAXIMUM,
00368                         input->variable[i].rangemax);
00369     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
00371                             LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
00374         DEFAULT_PRECISION)
00375         xml_node_set_uint (child, (const xmlChar *)
00376                             LABEL_PRECISION,
00377                             input->variable[i].precision);
00378     if (input->algorithm == ALGORITHM_SWEEP)
00379         xml_node_set_uint (child, (const xmlChar *)
00380                             LABEL_NSWEEPS,
00381                             input->variable[i].nsweeps);
00382     else if (input->algorithm == ALGORITHM_GENETIC)
00383         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00384                             input->variable[i].nbits);
00385     if (input->nsteps)
00386         xml_node_set_float (child, (const xmlChar *)
00387                             LABEL_STEP,
00388                             input->variable[i].step);
00389 }
00390 // Saving the error norm
00391 switch (input->norm)
00392 {
00393     case ERROR_NORM_MAXIMUM:
00394         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00395                     (const xmlChar *) LABEL_MAXIMUM);
00396         break;
00397     case ERROR_NORM_P:
00398         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00399                     (const xmlChar *) LABEL_P);
00400         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00401                             input->p);
00402         break;
00403     case ERROR_NORM_TAXICAB:
00404         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00405                     (const xmlChar *) LABEL_TAXICAB);
00406 }
00407 #if DEBUG_INTERFACE
00408 fprintf (stderr, "input_save: end\n");
00409 #endif
00410 }
00411 void
00412 input_save_json (JsonGenerator * generator)
00413 {
00414     unsigned int i, j;
00415     char *buffer;
00416     JsonNode *node, *child;
00417     JsonObject *object;
00418     JsonArray *array;
00419     GFile *file, *file2;

```

```

00420
00421 #if DEBUG_INTERFACE
00422     fprintf (stderr, "input_save_json: start\n");
00423 #endif
00424
00425     // Setting root JSON node
00426     node = json_node_new (JSON_NODE_OBJECT);
00427     object = json_node_get_object (node);
00428     json_generator_set_root (generator, node);
00429
00430     // Adding properties to the root JSON node
00431     if (strcmp (input->result, result_name))
00432         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00433     if (strcmp (input->variables, variables_name))
00434         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00435     file = g_file_new_for_path (input->directory);
00436     file2 = g_file_new_for_path (input->simulator);
00437     buffer = g_file_get_relative_path (file, file2);
00438     g_object_unref (file2);
00439     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00440     g_free (buffer);
00441     if (input->evaluator)
00442     {
00443         file2 = g_file_new_for_path (input->evaluator);
00444         buffer = g_file_get_relative_path (file, file2);
00445         g_object_unref (file2);
00446         if (strlen (buffer))
00447             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00448         g_free (buffer);
00449     }
00450     if (input->seed != DEFAULT_RANDOM_SEED)
00451         json_object_set_uint (object, LABEL_SEED,
input->seed);
00452
00453     // Setting the algorithm
00454     buffer = (char *) g_slice_alloc (64);
00455     switch (input->algorithm)
00456     {
00457     case ALGORITHM_MONTE_CARLO:
00458         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00459         snprintf (buffer, 64, "%u", input->nsimulations);
00460         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00461         snprintf (buffer, 64, "%u", input->niterations);
00462         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00463         snprintf (buffer, 64, "%.3lg", input->tolerance);
00464         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00465         snprintf (buffer, 64, "%u", input->nbest);
00466         json_object_set_string_member (object, LABEL_NBEST, buffer);
00467         input_save_direction_json (node);
00468         break;
00469     case ALGORITHM_SWEEP:
00470         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00471         snprintf (buffer, 64, "%u", input->niterations);
00472         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00473         snprintf (buffer, 64, "%.3lg", input->tolerance);
00474         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00475         snprintf (buffer, 64, "%u", input->nbest);
00476         json_object_set_string_member (object, LABEL_NBEST, buffer);
00477         input_save_direction_json (node);
00478         break;
00479     default:
00480         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00481         snprintf (buffer, 64, "%u", input->nsimulations);
00482         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00483         snprintf (buffer, 64, "%u", input->niterations);
00484         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00485         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00486         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00487         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00488         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00489         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00490         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00491         break;
00492     }
00493     g_slice_free1 (64, buffer);
00494     if (input->threshold != 0.)
00495         json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00496
00497     // Setting the experimental data
00498     array = json_array_new ();
00499     for (i = 0; i < input->nexperiments; ++i)

```

```

00502     {
00503         child = json_node_new (JSON_NODE_OBJECT);
00504         object = json_node_get_object (child);
00505         json_object_set_string_member (object, LABEL_NAME,
00506                                     input->experiment[i].name);
00507         if (input->experiment[i].weight != 1.)
00508             json_object_set_float (object, LABEL_WEIGHT,
00509                                   input->experiment[i].weight);
00510         for (j = 0; j < input->experiment->ninputs; ++j)
00511             json_object_set_string_member (object, stencil[j],
00512                                           input->experiment[i].
00513 stencil[j]);
00514         json_array_add_element (array, child);
00515     }
00516     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
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 (object, LABEL_NAME,
00524                                     input->variable[i].name);
00525         json_object_set_float (object, LABEL_MINIMUM,
00526                               input->variable[i].rangemin);
00527         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00528             json_object_set_float (object,
00529 LABEL_ABSOLUTE_MINIMUM,
00530                                   input->variable[i].rangeminabs);
00531         json_object_set_float (object, LABEL_MAXIMUM,
00532                               input->variable[i].rangemax);
00533         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00534             json_object_set_float (object,
00535 LABEL_ABSOLUTE_MAXIMUM,
00536                                   input->variable[i].rangemaxabs);
00537         if (input->variable[i].precision !=
00538 DEFAULT_PRECISION)
00539             json_object_set_uint (object, LABEL_PRECISION,
00540                                  input->variable[i].precision);
00541         if (input->algorithm == ALGORITHM_SWEEP)
00542             json_object_set_uint (object, LABEL_NSWEEPS,
00543                                  input->variable[i].nsweeps);
00544         else if (input->algorithm == ALGORITHM_GENETIC)
00545             json_object_set_uint (object, LABEL_NBITS,
00546                                  input->variable[i].nbits);
00547         if (input->nsteps)
00548             json_object_set_float (object, LABEL_STEP,
00549                                   input->variable[i].step);
00550         json_array_add_element (array, child);
00551     }
00552     json_object_set_array_member (object, LABEL_VARIABLES, array);
00553     // Saving the error norm
00554     switch (input->norm)
00555     {
00556     case ERROR_NORM_MAXIMUM:
00557         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00558         break;
00559     case ERROR_NORM_P:
00560         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00561         json_object_set_float (object, LABEL_P, input->
00562 p);
00563         break;
00564     case ERROR_NORM_TAXICAB:
00565         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00566     }
00567     }
00568     #if DEBUG_INTERFACE
00569     fprintf (stderr, "input_save_json: end\n");
00570     #endif
00571 }
00572 void
00573 input_save (char *filename)
00574 {
00575     xmlDoc *doc;
00576     JsonGenerator *generator;
00577     #if DEBUG_INTERFACE
00578     fprintf (stderr, "input_save: start\n");
00579     #endif
00580     // Getting the input file directory
00581     input->name = g_path_get_basename (filename);
00582     input->directory = g_path_get_dirname (filename);
00583 }

```



```

00588     if (input->type == INPUT_TYPE_XML)
00589     {
00590         // Opening the input file
00591         doc = xmlNewDoc ((const xmlChar *) "1.0");
00592         input_save_xml (doc);
00593
00594         // Saving the XML file
00595         xmlSaveFormatFile (filename, doc, 1);
00596
00597         // Freeing memory
00598         xmlFreeDoc (doc);
00599     }
00600     else
00601     {
00602         // Opening the input file
00603         generator = json_generator_new ();
00604         json_generator_set_pretty (generator, TRUE);
00605         input_save_json (generator);
00606
00607         // Saving the JSON file
00608         json_generator_to_file (generator, filename, NULL);
00609
00610         // Freeing memory
00611         g_object_unref (generator);
00612     }
00613
00614 #if DEBUG_INTERFACE
00615     fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }
00618
00623 void
00624 options_new ()
00625 {
00626 #if DEBUG_INTERFACE
00627     fprintf (stderr, "options_new: start\n");
00628 #endif
00629     options->label_seed = (GtkLabel *)
00630         gtk_label_new (_("Pseudo-random numbers generator seed"));
00631     options->spin_seed = (GtkSpinButton *)
00632         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00633     gtk_widget_set_tooltip_text
00634         (GTK_WIDGET (options->spin_seed),
00635          _("Seed to init the pseudo-random numbers generator"));
00636     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00637     options->label_threads = (GtkLabel *)
00638         gtk_label_new (_("Threads number for the stochastic algorithm"));
00639     options->spin_threads
00640         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00641     gtk_widget_set_tooltip_text
00642         (GTK_WIDGET (options->spin_threads),
00643          _("Number of threads to perform the calibration/optimization for "
00644            "the stochastic algorithm"));
00645     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00646     options->label_direction = (GtkLabel *)
00647         gtk_label_new (_("Threads number for the direction search method"));
00648     options->spin_direction =
00649         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00650     gtk_widget_set_tooltip_text (GTK_WIDGET (options->spin_direction),
00651         _("Number of threads to perform the calibration/optimization for "
00652           "the direction search method"));
00653     gtk_spin_button_set_value (options->spin_direction,
00654         (gdouble) nthreads_direction);
00655     options->grid = (GtkGrid *) gtk_grid_new ();
00656     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00657     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00658     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads), 0, 1,
00659         1, 1);
00660     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads), 1, 1, 1,
00661         1);
00662     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction), 0, 2,
00663         1, 1);
00664     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction), 1, 2,
00665         1, 1);
00666     gtk_widget_show_all (GTK_WIDGET (options->grid));
00667     options->dialog = (GtkDialog *)
00668         gtk_dialog_new_with_buttons (_("Options"),
00669         window->window,
00670         GTK_DIALOG_MODAL,
00671         _("_OK"), GTK_RESPONSE_OK,
00672         _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00673     gtk_container_add
00674         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00675         GTK_WIDGET (options->grid));
00676

```

```

00677     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00678     {
00679         input->seed
00680         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00681         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00682         nthreads_direction
00683         = gtk_spin_button_get_value_as_int (options->spin_direction);
00684     }
00685     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00686 #if DEBUG_INTERFACE
00687     fprintf (stderr, "options_new: end\n");
00688 #endif
00689 }
00690
00695 void
00696 running_new ()
00697 {
00698 #if DEBUG_INTERFACE
00699     fprintf (stderr, "running_new: start\n");
00700 #endif
00701     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00702     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00703     running->grid = (GtkGrid *) gtk_grid_new ();
00704     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00705     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00706     running->dialog = (GtkDialog *)
00707         gtk_dialog_new_with_buttons (_("Calculating"),
00708                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00709     gtk_container_add (GTK_CONTAINER
00710                       (gtk_dialog_get_content_area (running->dialog)),
00711                       GTK_WIDGET (running->grid));
00712     gtk_spinner_start (running->spinner);
00713     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00714 #if DEBUG_INTERFACE
00715     fprintf (stderr, "running_new: end\n");
00716 #endif
00717 }
00718
00724 unsigned int
00725 window_get_algorithm ()
00726 {
00727     unsigned int i;
00728 #if DEBUG_INTERFACE
00729     fprintf (stderr, "window_get_algorithm: start\n");
00730 #endif
00731     i = gtk_array_get_active (window->button_algorithm,
00732                             NALGORITHMS);
00733 #if DEBUG_INTERFACE
00734     fprintf (stderr, "window_get_algorithm: %u\n", i);
00735     fprintf (stderr, "window_get_algorithm: end\n");
00736 #endif
00737     return i;
00738 }
00744 unsigned int
00745 window_get_direction ()
00746 {
00747     unsigned int i;
00748 #if DEBUG_INTERFACE
00749     fprintf (stderr, "window_get_direction: start\n");
00750 #endif
00751     i = gtk_array_get_active (window->button_direction,
00752                             NDIRECTIONS);
00753 #if DEBUG_INTERFACE
00754     fprintf (stderr, "window_get_direction: %u\n", i);
00755     fprintf (stderr, "window_get_direction: end\n");
00756 #endif
00757     return i;
00758 }
00764 unsigned int
00765 window_get_norm ()
00766 {
00767     unsigned int i;
00768 #if DEBUG_INTERFACE
00769     fprintf (stderr, "window_get_norm: start\n");
00770 #endif
00771     i = gtk_array_get_active (window->button_norm,
00772                             NNORMS);
00773 #if DEBUG_INTERFACE
00774     fprintf (stderr, "window_get_norm: %u\n", i);
00775     fprintf (stderr, "window_get_norm: end\n");
00776 #endif
00777     return i;
00778 }
00783 void

```

```

00784 window_save_direction ()
00785 {
00786     #if DEBUG_INTERFACE
00787         fprintf (stderr, "window_save_direction: start\n");
00788     #endif
00789     if (gtk_toggle_button_get_active
00790         (GTK_TOGGLE_BUTTON (window->check_direction)))
00791     {
00792         input->nsteps = gtk_spin_button_get_value_as_int (window->
spin_steps);
00793         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
00794         switch (window_get_direction ())
00795         {
00796             case DIRECTION_METHOD_COORDINATES:
00797                 input->direction = DIRECTION_METHOD_COORDINATES;
00798                 break;
00799             default:
00800                 input->direction = DIRECTION_METHOD_RANDOM;
00801                 input->nestimates
00802                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00803         }
00804     }
00805     else
00806         input->nsteps = 0;
00807     #if DEBUG_INTERFACE
00808         fprintf (stderr, "window_save_direction: end\n");
00809     #endif
00810 }
00811
00817 int
00818 window_save ()
00819 {
00820     GtkFileChooserDialog *dlg;
00821     GtkFileFilter *filter1, *filter2;
00822     char *buffer;
00823
00824     #if DEBUG_INTERFACE
00825         fprintf (stderr, "window_save: start\n");
00826     #endif
00827
00828     // Opening the saving dialog
00829     dlg = (GtkFileChooserDialog *)
gtk_file_chooser_dialog_new (_("Save file"),
00830                             window->window,
00831                             GTK_FILE_CHOOSER_ACTION_SAVE,
00832                             _("_Cancel"), GTK_RESPONSE_CANCEL,
00833                             _("_OK"), GTK_RESPONSE_OK, NULL);
00834
00835     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00836     buffer = g_build_filename (input->directory, input->name, NULL);
00837     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00838     g_free (buffer);
00839
00840     // Adding XML filter
00841     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00842     gtk_file_filter_set_name (filter1, "XML");
00843     gtk_file_filter_add_pattern (filter1, "*.xml");
00844     gtk_file_filter_add_pattern (filter1, "*.XML");
00845     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00846
00847     // Adding JSON filter
00848     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00849     gtk_file_filter_set_name (filter2, "JSON");
00850     gtk_file_filter_add_pattern (filter2, "*.json");
00851     gtk_file_filter_add_pattern (filter2, "*.JSON");
00852     gtk_file_filter_add_pattern (filter2, "*.js");
00853     gtk_file_filter_add_pattern (filter2, "*.JS");
00854     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00855
00856     if (input->type == INPUT_TYPE_XML)
00857         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00858     else
00859         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861     // If OK response then saving
00862     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00863     {
00864         // Setting input file type
00865         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866         buffer = (char *) gtk_file_filter_get_name (filter1);
00867         if (!strcmp (buffer, "XML"))
00868             input->type = INPUT_TYPE_XML;
00869         else
00870             input->type = INPUT_TYPE_JSON;
00871
00872         // Adding properties to the root XML node
00873         input->simulator = gtk_file_chooser_get_filename

```

```

00874         (GTK_FILE_CHOOSER (window->button_simulator));
00875     if (gtk_toggle_button_get_active
00876         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00877         input->evaluator = gtk_file_chooser_get_filename
00878         (GTK_FILE_CHOOSER (window->button_evaluator));
00879     else
00880         input->evaluator = NULL;
00881     if (input->type == INPUT_TYPE_XML)
00882     {
00883         input->result
00884             = (char *) xmlStrdup ((const xmlChar *)
00885                                     gtk_entry_get_text (window->entry_result));
00886         input->variables
00887             = (char *) xmlStrdup ((const xmlChar *)
00888                                     gtk_entry_get_text (window->entry_variables));
00889     }
00890     else
00891     {
00892         input->result = g_strdup (gtk_entry_get_text (window->
00893 entry_result));
00894         input->variables =
00895             g_strdup (gtk_entry_get_text (window->entry_variables));
00896     }
00897     // Setting the algorithm
00898     switch (window_get_algorithm ())
00899     {
00900     case ALGORITHM_MONTE_CARLO:
00901         input->algorithm = ALGORITHM_MONTE_CARLO;
00902         input->nsimulations
00903             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00904         input->niterations
00905             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00906         input->tolerance = gtk_spin_button_get_value (window->
00907 spin_tolerance);
00908         input->nbest = gtk_spin_button_get_value_as_int (window->
00909 spin_bests);
00910         window_save_direction ();
00911         break;
00912     case ALGORITHM_SWEEP:
00913         input->algorithm = ALGORITHM_SWEEP;
00914         input->niterations
00915             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00916         input->tolerance = gtk_spin_button_get_value (window->
00917 spin_tolerance);
00918         input->nbest = gtk_spin_button_get_value_as_int (window->
00919 spin_bests);
00920         window_save_direction ();
00921         break;
00922     default:
00923         input->algorithm = ALGORITHM_GENETIC;
00924         input->nsimulations
00925             = gtk_spin_button_get_value_as_int (window->spin_population);
00926         input->niterations
00927             = gtk_spin_button_get_value_as_int (window->spin_generations);
00928         input->mutation_ratio
00929             = gtk_spin_button_get_value (window->spin_mutation);
00930         input->reproduction_ratio
00931             = gtk_spin_button_get_value (window->spin_reproduction);
00932         input->adaptation_ratio
00933             = gtk_spin_button_get_value (window->spin_adaptation);
00934         break;
00935     }
00936     input->norm = window_get_norm ();
00937     input->p = gtk_spin_button_get_value (window->spin_p);
00938     input->threshold = gtk_spin_button_get_value (window->
00939 spin_threshold);
00940
00941     // Saving the XML file
00942     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00943     input_save (buffer);
00944
00945     // Closing and freeing memory
00946     g_free (buffer);
00947     gtk_widget_destroy (GTK_WIDGET (dlg));
00948 #if DEBUG_INTERFACE
00949     fprintf (stderr, "window_save: end\n");
00950 #endif
00951     return 1;
00952 }
00953
00954 // Closing and freeing memory
00955 gtk_widget_destroy (GTK_WIDGET (dlg));
00956 #if DEBUG_INTERFACE
00957 fprintf (stderr, "window_save: end\n");
00958 #endif
00959 return 0;

```

```

00955 }
00956
00961 void
00962 window_run ()
00963 {
00964     unsigned int i;
00965     char *msg, *msg2, buffer[64], buffer2[64];
00966     #if DEBUG_INTERFACE
00967     fprintf (stderr, "window_run: start\n");
00968     #endif
00969     if (!window_save ())
00970     {
00971     #if DEBUG_INTERFACE
00972         fprintf (stderr, "window_run: end\n");
00973     #endif
00974         return;
00975     }
00976     running_new ();
00977     while (gtk_events_pending ())
00978         gtk_main_iteration ();
00979     optimize_open ();
00980     #if DEBUG_INTERFACE
00981     fprintf (stderr, "window_run: closing running dialog\n");
00982     #endif
00983     gtk_spinner_stop (running->spinner);
00984     gtk_widget_destroy (GTK_WIDGET (running->dialog));
00985     #if DEBUG_INTERFACE
00986     fprintf (stderr, "window_run: displaying results\n");
00987     #endif
00988     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00989     msg2 = g_strdup (buffer);
00990     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00991     {
00992         snprintf (buffer, 64, "%s = %s\n",
00993             input->variable[i].name, format[input->
00994             variable[i].precision]);
00995         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
00996         msg = g_strconcat (msg2, buffer2, NULL);
00997         g_free (msg2);
00998     }
00999     snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01000         optimize->calculation_time);
01001     msg = g_strconcat (msg2, buffer, NULL);
01002     g_free (msg2);
01003     show_message (_("Best result"), msg, INFO_TYPE);
01004     g_free (msg);
01005     #if DEBUG_INTERFACE
01006     fprintf (stderr, "window_run: freeing memory\n");
01007     #endif
01008     optimize_free ();
01009     #if DEBUG_INTERFACE
01010     fprintf (stderr, "window_run: end\n");
01011     #endif
01012 }
01013
01017 void
01018 window_help ()
01019 {
01020     char *buffer, *buffer2;
01021     #if DEBUG_INTERFACE
01022     fprintf (stderr, "window_help: start\n");
01023     #endif
01024     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01025         _("user-manual.pdf"), NULL);
01026     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01027     g_free (buffer2);
01028     #if GTK_MINOR_VERSION >= 22
01029     gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01030     #else
01031     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01032     #endif
01033     #if DEBUG_INTERFACE
01034     fprintf (stderr, "window_help: uri=%s\n", buffer);
01035     #endif
01036     g_free (buffer);
01037     #if DEBUG_INTERFACE
01038     fprintf (stderr, "window_help: end\n");
01039     #endif
01040 }
01041
01046 void
01047 window_about ()
01048 {
01049     static const gchar *authors[] = {
01050         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01051         "Borja Latorre Garcés <borja.latorre@csic.es>",
01052         NULL

```

```

01053     };
01054     #if DEBUG_INTERFACE
01055     fprintf (stderr, "window_about: start\n");
01056     #endif
01057     gtk_show_about_dialog
01058     (window->window,
01059      "program_name", "MPCOTool",
01060      "comments",
01061      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01062       "A software to perform calibrations or optimizations of empirical"
01063       " parameters"),
01064      "authors", authors,
01065      "translator-credits",
01066      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01067      "(english, french and spanish)\n"
01068      "Uğur Çayoğlu (german)",
01069      "version", "3.4.4",
01070      "copyright", "Copyright 2012-2017 Javier Burguete Tolosa",
01071      "logo", window->logo,
01072      "website", "https://github.com/jburguete/mpcotool",
01073      "license-type", GTK_LICENSE_BSD, NULL);
01074     #if DEBUG_INTERFACE
01075     fprintf (stderr, "window_about: end\n");
01076     #endif
01077 }
01078
01084 void
01085 window_update_direction ()
01086 {
01087     #if DEBUG_INTERFACE
01088     fprintf (stderr, "window_update_direction: start\n");
01089     #endif
01090     gtk_widget_show (GTK_WIDGET (window->check_direction));
01091     if (gtk_toggle_button_get_active
01092         (GTK_TOGGLE_BUTTON (window->check_direction)))
01093     {
01094         gtk_widget_show (GTK_WIDGET (window->grid_direction));
01095         gtk_widget_show (GTK_WIDGET (window->label_step));
01096         gtk_widget_show (GTK_WIDGET (window->spin_step));
01097     }
01098     switch (window_get_direction ())
01099     {
01100     case DIRECTION_METHOD_COORDINATES:
01101         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01102         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01103         break;
01104     default:
01105         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01106         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01107     }
01108     #if DEBUG_INTERFACE
01109     fprintf (stderr, "window_update_direction: end\n");
01110     #endif
01111 }
01112
01117 void
01118 window_update ()
01119 {
01120     unsigned int i;
01121     #if DEBUG_INTERFACE
01122     fprintf (stderr, "window_update: start\n");
01123     #endif
01124     gtk_widget_set_sensitive
01125     (GTK_WIDGET (window->button_evaluator),
01126      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01127                                   (window->check_evaluator)));
01128     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01129     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01130     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01131     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01132     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01133     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01134     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01135     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01136     gtk_widget_hide (GTK_WIDGET (window->label_population));
01137     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01138     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01139     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01140     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01142     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01144     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01146     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01148     gtk_widget_hide (GTK_WIDGET (window->label_bits));

```

```

01149 gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01150 gtk_widget_hide (GTK_WIDGET (window->check_direction));
01151 gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01152 gtk_widget_hide (GTK_WIDGET (window->label_step));
01153 gtk_widget_hide (GTK_WIDGET (window->spin_step));
01154 gtk_widget_hide (GTK_WIDGET (window->label_p));
01155 gtk_widget_hide (GTK_WIDGET (window->spin_p));
01156 i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01157 switch (window_get_algorithm ())
01158 {
01159     case ALGORITHM_MONTE_CARLO:
01160         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01161         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01162         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01163         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01164         if (i > 1)
01165         {
01166             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01167             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01168             gtk_widget_show (GTK_WIDGET (window->label_bests));
01169             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01170         }
01171         window_update_direction ();
01172         break;
01173     case ALGORITHM_SWEEP:
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         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01184         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01185         gtk_widget_show (GTK_WIDGET (window->check_direction));
01186         window_update_direction ();
01187         break;
01188     default:
01189         gtk_widget_show (GTK_WIDGET (window->label_population));
01190         gtk_widget_show (GTK_WIDGET (window->spin_population));
01191         gtk_widget_show (GTK_WIDGET (window->label_generations));
01192         gtk_widget_show (GTK_WIDGET (window->spin_generations));
01193         gtk_widget_show (GTK_WIDGET (window->label_mutation));
01194         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01195         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01196         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01197         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01198         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01199         gtk_widget_show (GTK_WIDGET (window->label_bits));
01200         gtk_widget_show (GTK_WIDGET (window->spin_bits));
01201     }
01202     gtk_widget_set_sensitive
01203     (GTK_WIDGET (window->button_remove_experiment),
01204      input->nexperiments > 1);
01204     gtk_widget_set_sensitive
01205     (GTK_WIDGET (window->button_remove_variable), input->
01206      nvariables > 1);
01206     for (i = 0; i < input->experiment->ninputs; ++i)
01207     {
01208         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01209         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01210         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01211         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01212         g_signal_handler_block
01213         (window->check_template[i], window->id_template[i]);
01214         g_signal_handler_block (window->button_template[i], window->
01215          id_input[i]);
01215         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01216          (window->check_template[i]), 1);
01217         g_signal_handler_unblock (window->button_template[i],
01218          window->id_input[i]);
01219         g_signal_handler_unblock (window->check_template[i],
01220          window->id_template[i]);
01221     }
01222     if (i > 0)
01223     {
01224         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01225         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01226          gtk_toggle_button_get_active
01227          GTK_TOGGLE_BUTTON (window->check_template
01228           [i - 1]));
01229     }
01230     if (i < MAX_NINPUTS)
01231     {
01232         gtk_widget_show (GTK_WIDGET (window->check_template[i]));

```

```

01233     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01234     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01235     gtk_widget_set_sensitive
01236         (GTK_WIDGET (window->button_template[i]),
01237          gtk_toggle_button_get_active
01238           (GTK_TOGGLE_BUTTON (window->check_template[i]));
01239     g_signal_handler_block
01240         (window->check_template[i], window->id_template[i]);
01241     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
01242     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                (window->check_template[i]), 0);
01243
01244     g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
01245     g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
01246
01247 }
01248
01249 while (++i < MAX_NINPUTS)
01250 {
01251     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01252     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01253 }
01254
01255 gtk_widget_set_sensitive
01256     (GTK_WIDGET (window->spin_minabs),
01257      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01258
01259 gtk_widget_set_sensitive
01260     (GTK_WIDGET (window->spin_maxabs),
01261      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01262
01263 if (window_get_norm () == ERROR_NORM_P)
01264 {
01265     gtk_widget_show (GTK_WIDGET (window->label_p));
01266     gtk_widget_show (GTK_WIDGET (window->spin_p));
01267 }
01268
01269 #if DEBUG_INTERFACE
01270 fprintf (stderr, "window_update: end\n");
01271 #endif
01272 }
01273
01274 void
01275 window_set_algorithm ()
01276 {
01277     int i;
01278
01279 #if DEBUG_INTERFACE
01280 fprintf (stderr, "window_set_algorithm: start\n");
01281 #endif
01282
01283 i = window_get_algorithm ();
01284
01285 switch (i)
01286 {
01287     case ALGORITHM_SWEEP:
01288         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01289         if (i < 0)
01290             i = 0;
01291         gtk_spin_button_set_value (window->spin_sweeps,
                                (gdouble) input->variable[i].
nsweeps);
01292         break;
01293     case ALGORITHM_GENETIC:
01294         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01295         if (i < 0)
01296             i = 0;
01297         gtk_spin_button_set_value (window->spin_bits,
                                (gdouble) input->variable[i].nbits);
01298     }
01299
01300 window_update ();
01301 #if DEBUG_INTERFACE
01302 fprintf (stderr, "window_set_algorithm: end\n");
01303 #endif
01304 }
01305
01306 void
01307 window_set_experiment ()
01308 {
01309     unsigned int i, j;
01310     char *buffer1, *buffer2;
01311
01312 #if DEBUG_INTERFACE
01313 fprintf (stderr, "window_set_experiment: start\n");
01314 #endif
01315
01316 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01317
01318 gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
01319
01320 buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01321
01322 buffer2 = g_build_filename (input->directory, buffer1, NULL);
01323
01324 g_free (buffer1);
01325
01326 g_signal_handler_block
01327     (window->button_experiment, window->id_experiment_name);
01328
01329 gtk_file_chooser_set_filename
01330     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);

```



```

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

```

```

01407     {
01408         input->experiment[j + 1].name
01409         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
name);
01410         for (j = 0; j < input->experiment->ninputs; ++j)
01411             input->experiment[i + 1].stencil[j]
01412             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
stencil[j]);
01413     }
01414     else
01415     {
01416         input->experiment[j + 1].name = g_strdup (input->
experiment[j].name);
01417         for (j = 0; j < input->experiment->ninputs; ++j)
01418             input->experiment[i + 1].stencil[j]
01419             = g_strdup (input->experiment[i].stencil[j]);
01420     }
01421     ++input->nexperiments;
01422     for (j = 0; j < input->experiment->ninputs; ++j)
01423         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01424     g_signal_handler_block
01425         (window->button_experiment, window->id_experiment_name);
01426     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01427     g_signal_handler_unblock
01428         (window->button_experiment, window->id_experiment_name);
01429     for (j = 0; j < input->experiment->ninputs; ++j)
01430         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
01431     window_update ();
01432 #if DEBUG_INTERFACE
01433     fprintf (stderr, "window_add_experiment: end\n");
01434 #endif
01435 }
01436
01441 void
01442 window_name_experiment ()
01443 {
01444     unsigned int i;
01445     char *buffer;
01446     GFile *file1, *file2;
01447 #if DEBUG_INTERFACE
01448     fprintf (stderr, "window_name_experiment: start\n");
01449 #endif
01450     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01451     file1
01452     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01453     file2 = g_file_new_for_path (input->directory);
01454     buffer = g_file_get_relative_path (file2, file1);
01455     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01456     gtk_combo_box_text_remove (window->combo_experiment, i);
01457     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01458     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01459     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01460     g_free (buffer);
01461     g_object_unref (file2);
01462     g_object_unref (file1);
01463 #if DEBUG_INTERFACE
01464     fprintf (stderr, "window_name_experiment: end\n");
01465 #endif
01466 }
01467
01472 void
01473 window_weight_experiment ()
01474 {
01475     unsigned int i;
01476 #if DEBUG_INTERFACE
01477     fprintf (stderr, "window_weight_experiment: start\n");
01478 #endif
01479     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01480     input->experiment[i].weight = gtk_spin_button_get_value (window->
spin_weight);
01481 #if DEBUG_INTERFACE
01482     fprintf (stderr, "window_weight_experiment: end\n");
01483 #endif
01484 }
01485
01491 void
01492 window_inputs_experiment ()
01493 {
01494     unsigned int j;
01495 #if DEBUG_INTERFACE
01496     fprintf (stderr, "window_inputs_experiment: start\n");
01497 #endif
01498     j = input->experiment->ninputs - 1;

```

```

01499     if (j
01500         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01501             (window->check_template[j])))
01502         --input->experiment->ninputs;
01503     if (input->experiment->ninputs < MAX_NINPUTS
01504         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01505             (window->check_template[j])))
01506         ++input->experiment->ninputs;
01507     window_update ();
01508 #if DEBUG_INTERFACE
01509     fprintf (stderr, "window_inputs_experiment: end\n");
01510 #endif
01511 }
01512
01520 void
01521 window_template_experiment (void *data)
01522 {
01523     unsigned int i, j;
01524     char *buffer;
01525     GFile *file1, *file2;
01526 #if DEBUG_INTERFACE
01527     fprintf (stderr, "window_template_experiment: start\n");
01528 #endif
01529     i = (size_t) data;
01530     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01531     file1
01532         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01533     file2 = g_file_new_for_path (input->directory);
01534     buffer = g_file_get_relative_path (file2, file1);
01535     if (input->type == INPUT_TYPE_XML)
01536         input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01537     else
01538         input->experiment[j].stencil[i] = g_strdup (buffer);
01539     g_free (buffer);
01540     g_object_unref (file2);
01541     g_object_unref (file1);
01542 #if DEBUG_INTERFACE
01543     fprintf (stderr, "window_template_experiment: end\n");
01544 #endif
01545 }
01546
01551 void
01552 window_set_variable ()
01553 {
01554     unsigned int i;
01555 #if DEBUG_INTERFACE
01556     fprintf (stderr, "window_set_variable: start\n");
01557 #endif
01558     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01559     g_signal_handler_block (window->entry_variable, window->
01560         id_variable_label);
01561     gtk_entry_set_text (window->entry_variable, input->variable[i].
01562         name);
01563     g_signal_handler_unblock (window->entry_variable, window->
01564         id_variable_label);
01565     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01566         rangemin);
01567     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01568         rangemax);
01569     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01570     {
01571         gtk_spin_button_set_value (window->spin_minabs,
01572             input->variable[i].rangeminabs);
01573         gtk_toggle_button_set_active
01574             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01575     }
01576     else
01577     {
01578         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01579         gtk_toggle_button_set_active
01580             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01581     }
01582     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01583     {
01584         gtk_spin_button_set_value (window->spin_maxabs,
01585             input->variable[i].rangemaxabs);
01586         gtk_toggle_button_set_active
01587             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01588     }
01589     else
01590     {
01591         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01592         gtk_toggle_button_set_active
01593             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01594     }
01595     gtk_spin_button_set_value (window->spin_precision,
01596         input->variable[i].precision);

```

```

01592     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
01593     if (input->nsteps)
01594         gtk_spin_button_set_value (window->spin_step, input->variable[i].
step);
01595     #if DEBUG_INTERFACE
01596         fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
input->variable[i].precision);
01597     #endif
01598     switch (window_get_algorithm ())
01599     {
01600     case ALGORITHM_SWEEP:
01601         gtk_spin_button_set_value (window->spin_sweeps,
(gdouble) input->variable[i].
nsweeps);
01602     #if DEBUG_INTERFACE
01603         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
input->variable[i].nsweeps);
01604     #endif
01605     case ALGORITHM_GENETIC:
01606         gtk_spin_button_set_value (window->spin_bits,
(gdouble) input->variable[i].nbits);
01607     #if DEBUG_INTERFACE
01608         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
input->variable[i].nbits);
01609     #endif
01610     break;
01611     }
01612     window_update ();
01613     #if DEBUG_INTERFACE
01614         fprintf (stderr, "window_set_variable: end\n");
01615     #endif
01616 }
01617 void
01618 window_remove_variable ()
01619 {
01620     unsigned int i, j;
01621     #if DEBUG_INTERFACE
01622         fprintf (stderr, "window_remove_variable: start\n");
01623     #endif
01624     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01625     g_signal_handler_block (window->combo_variable, window->
id_variable);
01626     gtk_combo_box_text_remove (window->combo_variable, i);
01627     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01628     xmlFree (input->variable[i].name);
01629     --input->nvariables;
01630     for (j = i; j < input->nvariables; ++j)
01631         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01632     j = input->nvariables - 1;
01633     if (i > j)
01634         i = j;
01635     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01636     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01637     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01638     window_update ();
01639     #if DEBUG_INTERFACE
01640         fprintf (stderr, "window_remove_variable: end\n");
01641     #endif
01642 }
01643 void
01644 window_add_variable ()
01645 {
01646     unsigned int i, j;
01647     #if DEBUG_INTERFACE
01648         fprintf (stderr, "window_add_variable: start\n");
01649     #endif
01650     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01651     g_signal_handler_block (window->combo_variable, window->
id_variable);
01652     gtk_combo_box_text_insert_text (window->combo_variable, i,
input->variable[i].name);
01653     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01654     input->variable = (Variable *) g_realloc
(input->variable, (input->nvariables + 1) * sizeof (
Variable));
01655     for (j = input->nvariables - 1; j > i; --j)
01656         memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));

```

```

01675     memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01676     if (input->type == INPUT_TYPE_XML)
01677         input->variable[j + 1].name
01678         = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01679     else
01680         input->variable[j + 1].name = g_strdup (input->
variable[j].name);
01681     ++input->nvariables;
01682     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01683     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01684     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01685     window_update ();
01686 #if DEBUG_INTERFACE
01687     fprintf (stderr, "window_add_variable: end\n");
01688 #endif
01689 }
01690
01695 void
01696 window_label_variable ()
01697 {
01698     unsigned int i;
01699     const char *buffer;
01700 #if DEBUG_INTERFACE
01701     fprintf (stderr, "window_label_variable: start\n");
01702 #endif
01703     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01704     buffer = gtk_entry_get_text (window->entry_variable);
01705     g_signal_handler_block (window->combo_variable, window->
id_variable_label);
01706     gtk_combo_box_text_remove (window->combo_variable, i);
01707     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01708     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01709     g_signal_handler_unblock (window->combo_variable, window->
id_variable_label);
01710 #if DEBUG_INTERFACE
01711     fprintf (stderr, "window_label_variable: end\n");
01712 #endif
01713 }
01714
01719 void
01720 window_precision_variable ()
01721 {
01722     unsigned int i;
01723 #if DEBUG_INTERFACE
01724     fprintf (stderr, "window_precision_variable: start\n");
01725 #endif
01726     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01727     input->variable[i].precision
01728     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01729     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
precision);
01730     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
precision);
01731     gtk_spin_button_set_digits (window->spin_minabs,
input->variable[i].precision);
01732     gtk_spin_button_set_digits (window->spin_maxabs,
input->variable[i].precision);
01733 #if DEBUG_INTERFACE
01734     fprintf (stderr, "window_precision_variable: end\n");
01735 #endif
01736 }
01737
01739 void
01740 window_rangemin_variable ()
01741 {
01742     unsigned int i;
01743 #if DEBUG_INTERFACE
01744     fprintf (stderr, "window_rangemin_variable: start\n");
01745 #endif
01746     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01747     input->variable[i].rangemin = gtk_spin_button_get_value (window->
spin_min);
01748 #if DEBUG_INTERFACE
01749     fprintf (stderr, "window_rangemin_variable: end\n");
01750 #endif
01751 }
01752
01757 void
01758 window_rangemax_variable ()
01759 {
01760     unsigned int i;
01761 #if DEBUG_INTERFACE
01762     fprintf (stderr, "window_rangemax_variable: start\n");
01763 #endif

```

```

01769 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01770 input->variable[i].rangemax = gtk_spin_button_get_value (window->
    spin_max);
01771 #if DEBUG_INTERFACE
01772 fprintf (stderr, "window_rangemax_variable: end\n");
01773 #endif
01774 }
01775
01780 void
01781 window_rangeminabs_variable ()
01782 {
01783     unsigned int i;
01784     #if DEBUG_INTERFACE
01785     fprintf (stderr, "window_rangeminabs_variable: start\n");
01786     #endif
01787     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01788     input->variable[i].rangeminabs
01789     = gtk_spin_button_get_value (window->spin_minabs);
01790     #if DEBUG_INTERFACE
01791     fprintf (stderr, "window_rangeminabs_variable: end\n");
01792     #endif
01793 }
01794
01799 void
01800 window_rangemaxabs_variable ()
01801 {
01802     unsigned int i;
01803     #if DEBUG_INTERFACE
01804     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01805     #endif
01806     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01807     input->variable[i].rangemaxabs
01808     = gtk_spin_button_get_value (window->spin_maxabs);
01809     #if DEBUG_INTERFACE
01810     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01811     #endif
01812 }
01813
01818 void
01819 window_step_variable ()
01820 {
01821     unsigned int i;
01822     #if DEBUG_INTERFACE
01823     fprintf (stderr, "window_step_variable: start\n");
01824     #endif
01825     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01826     input->variable[i].step = gtk_spin_button_get_value (window->
    spin_step);
01827     #if DEBUG_INTERFACE
01828     fprintf (stderr, "window_step_variable: end\n");
01829     #endif
01830 }
01831
01836 void
01837 window_update_variable ()
01838 {
01839     int i;
01840     #if DEBUG_INTERFACE
01841     fprintf (stderr, "window_update_variable: start\n");
01842     #endif
01843     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01844     if (i < 0)
01845         i = 0;
01846     switch (window_get_algorithm ())
01847     {
01848     case ALGORITHM_SWEEP:
01849         input->variable[i].nsweeps
01850         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01851     #if DEBUG_INTERFACE
01852         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01853             input->variable[i].nsweeps);
01854     #endif
01855         break;
01856     case ALGORITHM_GENETIC:
01857         input->variable[i].nbits
01858         = gtk_spin_button_get_value_as_int (window->spin_bits);
01859     #if DEBUG_INTERFACE
01860         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01861             input->variable[i].nbits);
01862     #endif
01863     }
01864     #if DEBUG_INTERFACE
01865     fprintf (stderr, "window_update_variable: end\n");
01866     #endif
01867 }
01868
01876 int

```

```

01877 window_read (char *filename)
01878 {
01879     unsigned int i;
01880     char *buffer;
01881     #if DEBUG_INTERFACE
01882     fprintf (stderr, "window_read: start\n");
01883     #endif
01884
01885     // Reading new input file
01886     input_free ();
01887     input->result = input->variables = NULL;
01888     if (!input_open (filename))
01889     {
01890     #if DEBUG_INTERFACE
01891         fprintf (stderr, "window_read: end\n");
01892     #endif
01893         return 0;
01894     }
01895
01896     // Setting GTK+ widgets data
01897     gtk_entry_set_text (window->entry_result, input->result);
01898     gtk_entry_set_text (window->entry_variables, input->
variables);
01899     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01900     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01901     g_free (buffer);
01902     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01903     if (input->evaluator)
01904     {
01905         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01906         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01907         g_free (buffer);
01908     }
01909     gtk_toggle_button_set_active
01910     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01911     switch (input->algorithm)
01912     {
01913     case ALGORITHM_MONTE_CARLO:
01914         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01915     case ALGORITHM_SWEEP:
01916         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01917         gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01918         gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01919         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_direction),
input->nsteps);
01920         if (input->nsteps)
01921         {
01922             gtk_toggle_button_set_active
01923             (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01924             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01925             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01926             switch (input->direction)
01927             {
01928             case DIRECTION_METHOD_RANDOM:
01929                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01930             }
01931         }
01932         break;
01933     default:
01934         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
01935         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
01936         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01937         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
01938         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
01939     }
01940     gtk_toggle_button_set_active
01941     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);

```

```

01956     gtk_spin_button_set_value (window->spin_p, input->p);
01957     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01958     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01959     g_signal_handler_block (window->button_experiment,
                                window->id_experiment_name);
01960     gtk_combo_box_text_remove_all (window->combo_experiment);
01961     for (i = 0; i < input->nexperiments; ++i)
01962         gtk_combo_box_text_append_text (window->combo_experiment,
                                input->experiment[i].name);
01963     g_signal_handler_unblock
01964         (window->button_experiment, window->id_experiment_name);
01965     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01966     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01967     g_signal_handler_block (window->combo_variable, window->
id_variable);
01970     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01971     gtk_combo_box_text_remove_all (window->combo_variable);
01972     for (i = 0; i < input->nvariables; ++i)
01973         gtk_combo_box_text_append_text (window->combo_variable,
                                input->variable[i].name);
01974     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01975     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01976     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01977     window_set_variable ();
01978     window_update ();
01979
01980     #if DEBUG_INTERFACE
01981     fprintf (stderr, "window_read: end\n");
01982     #endif
01983     return 1;
01984 }
01985
01986 void
01991 window_open ()
01992 {
01993     GtkFileChooserDialog *dlg;
01994     GtkFileFilter *filter;
01995     char *buffer, *directory, *name;
01996
01997     #if DEBUG_INTERFACE
01998     fprintf (stderr, "window_open: start\n");
01999     #endif
02000
02001     // Saving a backup of the current input file
02002     directory = g_strdup (input->directory);
02003     name = g_strdup (input->name);
02004
02005     // Opening dialog
02006     dlg = (GtkFileChooserDialog *)
02007         gtk_file_chooser_dialog_new (_("Open input file"),
                                window->window,
02008                                GTK_FILE_CHOOSER_ACTION_OPEN,
02009                                _("_Cancel"), GTK_RESPONSE_CANCEL,
02010                                _("_OK"), GTK_RESPONSE_OK, NULL);
02011
02012     // Adding XML filter
02013     filter = (GtkFileFilter *) gtk_file_filter_new ();
02014     gtk_file_filter_set_name (filter, "XML");
02015     gtk_file_filter_add_pattern (filter, "*.xml");
02016     gtk_file_filter_add_pattern (filter, "*.XML");
02017     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02018
02019     // Adding JSON filter
02020     filter = (GtkFileFilter *) gtk_file_filter_new ();
02021     gtk_file_filter_set_name (filter, "JSON");
02022     gtk_file_filter_add_pattern (filter, "*.json");
02023     gtk_file_filter_add_pattern (filter, "*.JSON");
02024     gtk_file_filter_add_pattern (filter, "*.js");
02025     gtk_file_filter_add_pattern (filter, "*.JS");
02026     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02027
02028     // If OK saving
02029     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02030     {
02031         // Trying to open the input file
02032         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02033         if (!window_read (buffer))
02034         {
02035             #if DEBUG_INTERFACE
02036             fprintf (stderr, "window_open: error reading input file\n");
02037             #endif
02038         }
02039     }

```



```

02040 #endif
02041     g_free (buffer);
02042
02043     // Reading backup file on error
02044     buffer = g_build_filename (directory, name, NULL);
02045     input->result = input->variables = NULL;
02046     if (!input_open (buffer))
02047     {
02048
02049         // Closing on backup file reading error
02050 #if DEBUG_INTERFACE
02051         fprintf (stderr, "window_read: error reading backup file\n");
02052 #endif
02053         g_free (buffer);
02054         break;
02055     }
02056     g_free (buffer);
02057 }
02058 else
02059 {
02060     g_free (buffer);
02061     break;
02062 }
02063 }
02064
02065 // Freeing and closing
02066 g_free (name);
02067 g_free (directory);
02068 gtk_widget_destroy (GTK_WIDGET (dlg));
02069 #if DEBUG_INTERFACE
02070 fprintf (stderr, "window_open: end\n");
02071 #endif
02072 }
02073
02080 void
02081 window_new (GtkApplication * application)
02082 {
02083     unsigned int i;
02084     char *buffer, *buffer2, buffer3[64];
02085     char *label_algorithm[NALGORITHMS] = {
02086         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02087     };
02088     char *tip_algorithm[NALGORITHMS] = {
02089         _("Monte-Carlo brute force algorithm"),
02090         _("Sweep brute force algorithm"),
02091         _("Genetic algorithm")
02092     };
02093     char *label_direction[N DIRECTIONS] = {
02094         _("_Coordinates descent"), _("_Random")
02095     };
02096     char *tip_direction[N DIRECTIONS] = {
02097         _("Coordinates direction estimate method"),
02098         _("Random direction estimate method")
02099     };
02100     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02101     char *tip_norm[NNORMS] = {
02102         _("Euclidean error norm (L2)"),
02103         _("Maximum error norm (L)"),
02104         _("P error norm (Lp)"),
02105         _("Taxicab error norm (L1)")
02106     };
02107
02108 #if DEBUG_INTERFACE
02109     fprintf (stderr, "window_new: start\n");
02110 #endif
02111
02112     // Creating the window
02113     window->window = main_window
02114         = (GtkWindow *) gtk_application_window_new (application);
02115
02116     // Finish when closing the window
02117     g_signal_connect_swapped (window->window, "delete-event",
02118                             G_CALLBACK (g_application_quit),
02119                             G_APPLICATION (application));
02120
02121     // Setting the window title
02122     gtk_window_set_title (window->window, "MPCOTool");
02123
02124     // Creating the open button
02125     window->button_open = (GtkToolButton *) gtk_tool_button_new
02126         (gtk_image_new_from_icon_name ("document-open",
02127                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02128     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02129
02130     // Creating the save button
02131     window->button_save = (GtkToolButton *) gtk_tool_button_new
02132         (gtk_image_new_from_icon_name ("document-save",

```

```

02133                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02134 g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
02135                                     NULL);
02136
02137 // Creating the run button
02138 window->button_run = (GtkToolButton *) gtk_tool_button_new
02139     (gtk_image_new_from_icon_name ("system-run",
02140                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02141 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02142
02143 // Creating the options button
02144 window->button_options = (GtkToolButton *) gtk_tool_button_new
02145     (gtk_image_new_from_icon_name ("preferences-system",
02146                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02147 g_signal_connect (window->button_options, "clicked", options_new, NULL);
02148
02149 // Creating the help button
02150 window->button_help = (GtkToolButton *) gtk_tool_button_new
02151     (gtk_image_new_from_icon_name ("help-browser",
02152                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02153 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02154
02155 // Creating the about button
02156 window->button_about = (GtkToolButton *) gtk_tool_button_new
02157     (gtk_image_new_from_icon_name ("help-about",
02158                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02159 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02160
02161 // Creating the exit button
02162 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02163     (gtk_image_new_from_icon_name ("application-exit",
02164                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02165 g_signal_connect_swapped (window->button_exit, "clicked",
02166                           G_CALLBACK (g_application_quit),
02167                           G_APPLICATION (application));
02168
02169 // Creating the buttons bar
02170 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02171 gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02173 gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02175 gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02177 gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02179 gtk_toolbar_insert
02180     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02181 gtk_toolbar_insert
02182     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02183 gtk_toolbar_insert
02184     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02185 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02186
02187 // Creating the simulator program label and entry
02188 window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02189 window->button_simulator = (GtkFileChooserButton *)
02190     gtk_file_chooser_button_new (_("Simulator program"),
02191                                   GTK_FILE_CHOOSER_ACTION_OPEN);
02192 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02193                               _("Simulator program executable file"));
02194 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02195
02196 // Creating the evaluator program label and entry
02197 window->check_evaluator = (GtkCheckButton *)
02198     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02199 g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
02200 window->button_evaluator = (GtkFileChooserButton *)
02201     gtk_file_chooser_button_new (_("Evaluator program"),
02202                                   GTK_FILE_CHOOSER_ACTION_OPEN);
02203 gtk_widget_set_tooltip_text
02204     (GTK_WIDGET (window->button_evaluator),
02205      _("Optional evaluator program executable file"));
02206
02207 // Creating the results files labels and entries
02208 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02209 window->entry_result = (GtkEntry *) gtk_entry_new ();
02210 gtk_widget_set_tooltip_text
02211     (GTK_WIDGET (window->entry_result), _("Best results file"));
02212 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02213 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02214 gtk_widget_set_tooltip_text
02215     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02216
02217 // Creating the files grid and attaching widgets

```

```

02218     window->grid_files = (GtkGrid *) gtk_grid_new ();
02219     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02220                     0, 0, 1, 1);
02221     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02222                     1, 0, 1, 1);
02223     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02224                     0, 1, 1, 1);
02225     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02226                     1, 1, 1, 1);
02227     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02228                     0, 2, 1, 1);
02229     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02230                     1, 2, 1, 1);
02231     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02232                     0, 3, 1, 1);
02233     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02234                     1, 3, 1, 1);
02235
02236     // Creating the algorithm properties
02237     window->label_simulations = (GtkLabel *) gtk_label_new
02238     (_("Simulations number"));
02239     window->spin_simulations
02240     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02241     gtk_widget_set_tooltip_text
02242     (GTK_WIDGET (window->spin_simulations),
02243      _("Number of simulations to perform for each iteration"));
02244     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02245     window->label_iterations = (GtkLabel *)
02246     gtk_label_new (_("Iterations number"));
02247     window->spin_iterations
02248     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02249     gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02251     g_signal_connect
02252     (window->spin_iterations, "value-changed", window_update, NULL);
02253     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02254     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02255     window->spin_tolerance =
02256     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02257     gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_tolerance),
02259      _("Tolerance to set the variable interval on the next iteration"));
02260     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02261     window->spin_bests
02262     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02263     gtk_widget_set_tooltip_text
02264     (GTK_WIDGET (window->spin_bests),
02265      _("Number of best simulations used to set the variable interval "
02266        "on the next iteration"));
02267     window->label_population
02268     = (GtkLabel *) gtk_label_new (_("Population number"));
02269     window->spin_population
02270     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02271     gtk_widget_set_tooltip_text
02272     (GTK_WIDGET (window->spin_population),
02273      _("Number of population for the genetic algorithm"));
02274     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02275     window->label_generations
02276     = (GtkLabel *) gtk_label_new (_("Generations number"));
02277     window->spin_generations
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02279     gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_generations),
02281      _("Number of generations for the genetic algorithm"));
02282     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02283     window->spin_mutation
02284     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02285     gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_mutation),
02287      _("Ratio of mutation for the genetic algorithm"));
02288     window->label_reproduction
02289     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02290     window->spin_reproduction
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292     gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_reproduction),
02294      _("Ratio of reproduction for the genetic algorithm"));
02295     window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02296     window->spin_adaptation

```

```

02297     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02298     gtk_widget_set_tooltip_text
02299         (GTK_WIDGET (window->spin_adaptation),
02300          _("Ratio of adaptation for the genetic algorithm"));
02301     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02302     window->spin_threshold = (GtkSpinButton *)
02303         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02304                                         precision[DEFAULT_PRECISION]);
02305     gtk_widget_set_tooltip_text
02306         (GTK_WIDGET (window->spin_threshold),
02307          _("Threshold in the objective function to finish the simulations"));
02308     window->scrolled_threshold =
02309         (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02310     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02311                       GTK_WIDGET (window->spin_threshold));
02312     // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02313     // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02314     //                         GTK_ALIGN_FILL);
02315
02316     // Creating the direction search method properties
02317     window->check_direction = (GtkCheckButton *)
02318         gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02319     g_signal_connect (window->check_direction, "clicked",
02320                      window_update, NULL);
02321     window->grid_direction = (GtkGrid *) gtk_grid_new ();
02322     window->button_direction[0] = (GtkRadioButton *)
02323         gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02324     gtk_grid_attach (window->grid_direction,
02325                     GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02326     g_signal_connect (window->button_direction[0], "clicked",
02327                      window_update, NULL);
02328     for (i = 0; ++i < NDIRECTIONS;)
02329     {
02330         window->button_direction[i] = (GtkRadioButton *)
02331             gtk_radio_button_new_with_mnemonic
02332             (gtk_radio_button_get_group (window->button_direction[0]),
02333              label_direction[i]);
02334         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02335                                     tip_direction[i]);
02336         gtk_grid_attach (window->grid_direction,
02337                         GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02338         g_signal_connect (window->button_direction[i], "clicked",
02339                          window_update, NULL);
02340     }
02341     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02342     window->spin_steps = (GtkSpinButton *)
02343         gtk_spin_button_new_with_range (1., 1.e12, 1.);
02344     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02345     window->label_estimates
02346         = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02347     window->spin_estimates = (GtkSpinButton *)
02348         gtk_spin_button_new_with_range (1., 1.e3, 1.);
02349     window->label_relaxation
02350         = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02351     window->spin_relaxation = (GtkSpinButton *)
02352         gtk_spin_button_new_with_range (0., 2., 0.001);
02353     gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02354 label_steps),
02355                     0, NDIRECTIONS, 1, 1);
02356     gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02357 spin_steps),
02358                     1, NDIRECTIONS, 1, 1);
02359     gtk_grid_attach (window->grid_direction,
02360                     GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02361                     1, 1);
02362     gtk_grid_attach (window->grid_direction,
02363                     GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02364                     1);
02365     gtk_grid_attach (window->grid_direction,
02366                     GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02367                     1, 1);
02368     gtk_grid_attach (window->grid_direction,
02369                     GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02370                     1, 1);
02371
02372     // Creating the array of algorithms
02373     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02374     window->button_algorithm[0] = (GtkRadioButton *)
02375         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02376     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02377                                 tip_algorithm[0]);
02378     gtk_grid_attach (window->grid_algorithm,
02379                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02380     g_signal_connect (window->button_algorithm[0], "clicked",
02381                      window_set_algorithm, NULL);
02382     for (i = 0; ++i < NALGORITHMS;)

```

```

02380     {
02381         window->button_algorithm[i] = (GtkRadioButton *)
02382             gtk_radio_button_new_with_mnemonic
02383             (gtk_radio_button_get_group (window->button_algorithm[0]),
02384              label_algorithm[i]);
02385         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02386                                     tip_algorithm[i]);
02387         gtk_grid_attach (window->grid_algorithm,
02388                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02389         g_signal_connect (window->button_algorithm[i], "clicked",
02390                          window_set_algorithm, NULL);
02391     }
02392     gtk_grid_attach (window->grid_algorithm,
02393                     GTK_WIDGET (window->label_simulations), 0,
02394                     NALGORITHMS, 1, 1);
02395     gtk_grid_attach (window->grid_algorithm,
02396                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02397     gtk_grid_attach (window->grid_algorithm,
02398                     GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02399                     1, 1);
02400     gtk_grid_attach (window->grid_algorithm,
02401                     GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02402                     1, 1);
02403     gtk_grid_attach (window->grid_algorithm,
02404                     GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02405                     1, 1);
02406     gtk_grid_attach (window->grid_algorithm,
02407                     GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02408                     1);
02409     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
02410 label_bests),
02411                     0, NALGORITHMS + 3, 1, 1);
02412     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
02413 spin_bests), 1,
02414                     NALGORITHMS + 3, 1, 1);
02415     gtk_grid_attach (window->grid_algorithm,
02416                     GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02417                     1, 1);
02418     gtk_grid_attach (window->grid_algorithm,
02419                     GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02420                     1, 1);
02421     gtk_grid_attach (window->grid_algorithm,
02422                     GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02423                     1, 1);
02424     gtk_grid_attach (window->grid_algorithm,
02425                     GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02426                     1, 1);
02427     gtk_grid_attach (window->grid_algorithm,
02428                     GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02429                     1);
02430     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
02431 spin_mutation),
02432                     1, NALGORITHMS + 6, 1, 1);
02433     gtk_grid_attach (window->grid_algorithm,
02434                     GTK_WIDGET (window->label_reproduction), 0,
02435                     NALGORITHMS + 7, 1, 1);
02436     gtk_grid_attach (window->grid_algorithm,
02437                     GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02438                     1, 1);
02439     gtk_grid_attach (window->grid_algorithm,
02440                     GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02441                     1, 1);
02442     gtk_grid_attach (window->grid_algorithm,
02443                     GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02444                     1, 1);
02445     gtk_grid_attach (window->grid_algorithm,
02446                     GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02447                     2, 1);
02448     gtk_grid_attach (window->grid_algorithm,
02449                     GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02450                     2, 1);
02451     gtk_grid_attach (window->grid_algorithm,
02452                     GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02453                     1, 1);
02454     gtk_grid_attach (window->grid_algorithm,
02455                     GTK_WIDGET (window->scrolled_threshold), 1,
02456                     NALGORITHMS + 11, 1, 1);
02457     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02458     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02459                       GTK_WIDGET (window->grid_algorithm));
02460     // Creating the variable widgets
02461     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02462     gtk_widget_set_tooltip_text
02463         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02464     window->id_variable = g_signal_connect
02465         (window->combo_variable, "changed", window_set_variable, NULL);

```

```

02464 window->button_add_variable
02465     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02466     GTK_ICON_SIZE_BUTTON);
02467 g_signal_connect
02468     (window->button_add_variable, "clicked",
02469 window_add_variable, NULL);
02469 gtk_widget_set_tooltip_text
02470     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02471 window->button_remove_variable
02472     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02473     GTK_ICON_SIZE_BUTTON);
02474 g_signal_connect
02475     (window->button_remove_variable, "clicked",
02476 window_remove_variable, NULL);
02476 gtk_widget_set_tooltip_text
02477     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02478 window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02479 window->entry_variable = (GtkEntry *) gtk_entry_new ();
02480 gtk_widget_set_tooltip_text
02481     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02482 gtk_widget_set_expand (GTK_WIDGET (window->entry_variable), TRUE);
02483 window->id_variable_label = g_signal_connect
02484     (window->entry_variable, "changed", window_label_variable, NULL);
02485 window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02486 window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02487     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02488 gtk_widget_set_tooltip_text
02489     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02490 window->scrolled_min
02491     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02492 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02493     GTK_WIDGET (window->spin_min));
02494 g_signal_connect (window->spin_min, "value-changed",
02495     window_rangemin_variable, NULL);
02496 window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02497 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02498     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02499 gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02501 window->scrolled_max
02502     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02503 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02504     GTK_WIDGET (window->spin_max));
02505 g_signal_connect (window->spin_max, "value-changed",
02506     window_rangemax_variable, NULL);
02507 window->check_minabs = (GtkCheckButton *)
02508     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02509 g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02510 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02511     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02512 gtk_widget_set_tooltip_text
02513     (GTK_WIDGET (window->spin_minabs),
02514     _("Minimum allowed value of the variable"));
02515 window->scrolled_minabs
02516     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02517 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02518     GTK_WIDGET (window->spin_minabs));
02519 g_signal_connect (window->spin_minabs, "value-changed",
02520     window_rangeminabs_variable, NULL);
02521 window->check_maxabs = (GtkCheckButton *)
02522     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02523 g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02524 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02525     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02526 gtk_widget_set_tooltip_text
02527     (GTK_WIDGET (window->spin_maxabs),
02528     _("Maximum allowed value of the variable"));
02529 window->scrolled_maxabs
02530     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02531 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02532     GTK_WIDGET (window->spin_maxabs));
02533 g_signal_connect (window->spin_maxabs, "value-changed",
02534     window_rangemaxabs_variable, NULL);
02535 window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02536 window->spin_precision = (GtkSpinButton *)
02537     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02538 gtk_widget_set_tooltip_text
02539     (GTK_WIDGET (window->spin_precision),
02540     _("Number of precision floating point digits\n"
02541     "0 is for integer numbers"));
02542 g_signal_connect (window->spin_precision, "value-changed",
02543     window_precision_variable, NULL);
02544 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02545 window->spin_sweeps =
02546     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02547 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02548     _("Number of steps sweeping the variable"));

```

```

02549 g_signal_connect (window->spin_sweeps, "value-changed",
02550                   window_update_variable, NULL);
02551 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02552 window->spin_bits
02553     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02554 gtk_widget_set_tooltip_text
02555     (GTK_WIDGET (window->spin_bits),
02556      _("Number of bits to encode the variable"));
02557 g_signal_connect
02558     (window->spin_bits, "value-changed", window_update_variable, NULL);
02559 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02560 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02561     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02562 gtk_widget_set_tooltip_text
02563     (GTK_WIDGET (window->spin_step),
02564      _("Initial step size for the direction search method"));
02565 window->scrolled_step
02566     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02567 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02568                   GTK_WIDGET (window->spin_step));
02569 g_signal_connect
02570     (window->spin_step, "value-changed", window_step_variable, NULL);
02571 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02572 gtk_grid_attach (window->grid_variable,
02573                 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02574 gtk_grid_attach (window->grid_variable,
02575                 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02576 gtk_grid_attach (window->grid_variable,
02577                 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02578 gtk_grid_attach (window->grid_variable,
02579                 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02580 gtk_grid_attach (window->grid_variable,
02581                 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02582 gtk_grid_attach (window->grid_variable,
02583                 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02584 gtk_grid_attach (window->grid_variable,
02585                 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02586 gtk_grid_attach (window->grid_variable,
02587                 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02588 gtk_grid_attach (window->grid_variable,
02589                 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02590 gtk_grid_attach (window->grid_variable,
02591                 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02592 gtk_grid_attach (window->grid_variable,
02593                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02594 gtk_grid_attach (window->grid_variable,
02595                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02596 gtk_grid_attach (window->grid_variable,
02597                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02598 gtk_grid_attach (window->grid_variable,
02599                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02600 gtk_grid_attach (window->grid_variable,
02601                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02602 gtk_grid_attach (window->grid_variable,
02603                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02604 gtk_grid_attach (window->grid_variable,
02605                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02606 gtk_grid_attach (window->grid_variable,
02607                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02608 gtk_grid_attach (window->grid_variable,
02609                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02610 gtk_grid_attach (window->grid_variable,
02611                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02614 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02615 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02616                   GTK_WIDGET (window->grid_variable));
02617
02618 // Creating the experiment widgets
02619 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02620 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02621                             _("Experiment selector"));
02622 window->id_experiment = g_signal_connect
02623     (window->combo_experiment, "changed", window_set_experiment, NULL);
02624
02625 window->button_add_experiment
02626     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02627                                                   GTK_ICON_SIZE_BUTTON);
02628 g_signal_connect
02629     (window->button_add_experiment, "clicked",
02630      window_add_experiment, NULL);
02631 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02632                             _("Add experiment"));
02633 window->button_remove_experiment
02634     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02635                                                   GTK_ICON_SIZE_BUTTON);

```



```

02634 g_signal_connect (window->button_remove_experiment, "clicked",
02635                     window_remove_experiment, NULL);
02636 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02637                               _("Remove experiment"));
02638 window->label_experiment
02639     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02640 window->button_experiment = (GtkFileChooserButton *)
02641     gtk_file_chooser_button_new (_("Experimental data file"),
02642                                   GTK_FILE_CHOOSER_ACTION_OPEN);
02643 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02644                               _("Experimental data file"));
02645 window->id_experiment_name
02646     = g_signal_connect (window->button_experiment, "selection-changed",
02647                         window_name_experiment, NULL);
02648 gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02649 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02650 window->spin_weight
02651     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02652 gtk_widget_set_tooltip_text
02653     (GTK_WIDGET (window->spin_weight),
02654      _("Weight factor to build the objective function"));
02655 g_signal_connect
02656     (window->spin_weight, "value-changed", window_weight_experiment,
02657      NULL);
02657 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02658 gtk_grid_attach (window->grid_experiment,
02659                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02660 gtk_grid_attach (window->grid_experiment,
02661                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02662 gtk_grid_attach (window->grid_experiment,
02663                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02664 gtk_grid_attach (window->grid_experiment,
02665                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02666 gtk_grid_attach (window->grid_experiment,
02667                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02668 gtk_grid_attach (window->grid_experiment,
02669                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02670 gtk_grid_attach (window->grid_experiment,
02671                 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02672 for (i = 0; i < MAX_NINPUTS; ++i)
02673 {
02674     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02675     window->check_template[i] = (GtkCheckButton *)
02676     gtk_check_button_new_with_label (buffer3);
02677     window->id_template[i]
02678     = g_signal_connect (window->check_template[i], "toggled",
02679                         window_inputs_experiment, NULL);
02680     gtk_grid_attach (window->grid_experiment,
02681                     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02682     window->button_template[i] =
02683     (GtkFileChooserButton *)
02684     gtk_file_chooser_button_new (_("Input template"),
02685                                   GTK_FILE_CHOOSER_ACTION_OPEN);
02686     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02687                                   _("Experimental input template file"));
02688     window->id_input[i] =
02689     g_signal_connect_swapped (window->button_template[i],
02690                               "selection-changed",
02691                               (GCallback) window_template_experiment,
02692                               (void *) (size_t) i);
02693     gtk_grid_attach (window->grid_experiment,
02694                     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02695 }
02696 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02697 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02698                   GTK_WIDGET (window->grid_experiment));
02699
02700 // Creating the error norm widgets
02701 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02702 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02703 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02704                   GTK_WIDGET (window->grid_norm));
02705 window->button_norm[0] = (GtkRadioButton *)
02706     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02707 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02708                               tip_norm[0]);
02709 gtk_grid_attach (window->grid_norm,
02710                 GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02711 g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02712 for (i = 0; ++i < NNORMS;)
02713 {
02714     window->button_norm[i] = (GtkRadioButton *)
02715     gtk_radio_button_new_with_mnemonic
02716     (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02717     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02718                                   tip_norm[i]);
02719     gtk_grid_attach (window->grid_norm,

```



```

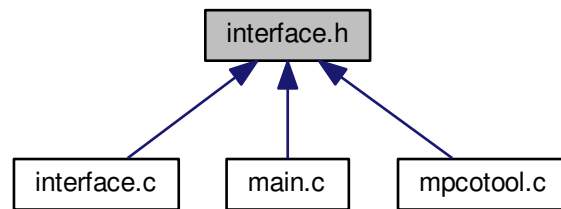
02720         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02721     g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
02722 }
02723 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02724 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02725 window->spin_p =
02726     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02727         G_MAXDOUBLE, 0.01);
02728 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02729     _("P parameter for the P error norm"));
02730 window->scrolled_p =
02731     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02732 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02733     GTK_WIDGET (window->spin_p));
02734 gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02735 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02736 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02737     1, 2, 1, 2);
02738
02739 // Creating the grid and attaching the widgets to the grid
02740 window->grid = (GtkGrid *) gtk_grid_new ();
02741 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02742 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02743 gtk_grid_attach (window->grid,
02744     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02745 gtk_grid_attach (window->grid,
02746     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02747 gtk_grid_attach (window->grid,
02748     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02749 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02750 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));
02751
02752 // Setting the window logo
02753 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02754 gtk_window_set_icon (window->window, window->logo);
02755
02756 // Showing the window
02757 gtk_widget_show_all (GTK_WIDGET (window->window));
02758
02759 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02760 #if GTK_MINOR_VERSION >= 16
02761     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02762     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02763     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02764     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02765     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02766     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02767     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02768 #endif
02769
02770 // Reading initial example
02771 input_new ();
02772 buffer2 = g_get_current_dir ();
02773 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02774 g_free (buffer2);
02775 window_read (buffer);
02776 g_free (buffer);
02777
02778 #if DEBUG_INTERFACE
02779     fprintf (stderr, "window_new: start\n");
02780 #endif
02781 }

```

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 2012-2017, 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 (  
    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 [566](#) of file [utils.c](#).

```

00567 {
00568     unsigned int i;
00569     for (i = 0; i < n; ++i)
00570         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00571             break;
00572     return i;
00573 }

```

4.13.2.2 input_save()

```

void input_save (
    char * filename )

```

Function to save the input file.

Parameters

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

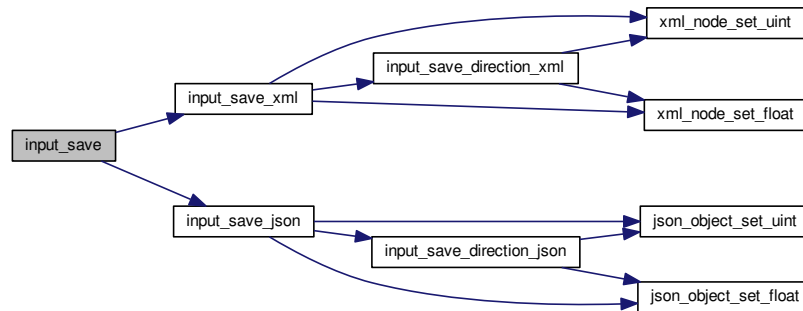
Definition at line 575 of file [interface.c](#).

```

00576 {
00577     xmlDoc *doc;
00578     JsonGenerator *generator;
00579
00580     #if DEBUG_INTERFACE
00581         fprintf (stderr, "input_save: start\n");
00582     #endif
00583
00584     // Getting the input file directory
00585     input->name = g_path_get_basename (filename);
00586     input->directory = g_path_get_dirname (filename);
00587
00588     if (input->type == INPUT_TYPE_XML)
00589     {
00590         // Opening the input file
00591         doc = xmlNewDoc ((const xmlChar *) "1.0");
00592         input_save_xml (doc);
00593
00594         // Saving the XML file
00595         xmlSaveFormatFile (filename, doc, 1);
00596
00597         // Freeing memory
00598         xmlFreeDoc (doc);
00599     }
00600     else
00601     {
00602         // Opening the input file
00603         generator = json_generator_new ();
00604         json_generator_set_pretty (generator, TRUE);
00605         input_save_json (generator);
00606
00607         // Saving the JSON file
00608         json_generator_to_file (generator, filename, NULL);
00609
00610         // Freeing memory
00611         g_object_unref (generator);
00612     }
00613
00614     #if DEBUG_INTERFACE
00615         fprintf (stderr, "input_save: end\n");
00616     #endif
00617 }

```

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

```

00726 {
00727     unsigned int i;
00728     #if DEBUG_INTERFACE
00729     fprintf (stderr, "window_get_algorithm: start\n");
00730     #endif
00731     i = gtk_array_get_active (window->button_algorithm,
00732                             NALGORITHMS);
00733     #if DEBUG_INTERFACE
00734     fprintf (stderr, "window_get_algorithm: %u\n", i);
00735     fprintf (stderr, "window_get_algorithm: end\n");
00736     #endif
00737     return i;
00738 }

```

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

```
00746 {  
00747     unsigned int i;  
00748     #if DEBUG_INTERFACE  
00749     fprintf (stderr, "window_get_direction: start\n");  
00750     #endif  
00751     i = gtk_array_get_active (window->button_direction,  
                             NDIRECTIONS);  
00752     #if DEBUG_INTERFACE  
00753     fprintf (stderr, "window_get_direction: %u\n", i);  
00754     fprintf (stderr, "window_get_direction: end\n");  
00755     #endif  
00756     return i;  
00757 }
```

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

```

00766 {
00767     unsigned int i;
00768     #if DEBUG_INTERFACE
00769     fprintf (stderr, "window_get_norm: start\n");
00770     #endif
00771     i = gtk_array_get_active (window->button_norm,
NNORMS);
00772     #if DEBUG_INTERFACE
00773     fprintf (stderr, "window_get_norm: %u\n", i);
00774     fprintf (stderr, "window_get_norm: end\n");
00775     #endif
00776     return i;
00777 }

```

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 2081 of file [interface.c](#).

```

02082 {
02083     unsigned int i;
02084     char *buffer, *buffer2, buffer3[64];
02085     char *label_algorithm[NALGORITHMS] = {
02086         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02087     };
02088     char *tip_algorithm[NALGORITHMS] = {
02089         _("Monte-Carlo brute force algorithm"),
02090         _("Sweep brute force algorithm"),
02091         _("Genetic algorithm")
02092     };
02093     char *label_direction[NDIRECTIONS] = {
02094         _("_Coordinates descent"), _("_Random")
02095     };
02096     char *tip_direction[NDIRECTIONS] = {
02097         _("Coordinates direction estimate method"),
02098         _("Random direction estimate method")
02099     };
02100     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02101     char *tip_norm[NNORMS] = {
02102         _("Euclidean error norm (L2)"),
02103         _("Maximum error norm (L)"),

```



```

02104     _("P error norm (Lp)"),
02105     _("Taxicab error norm (L1)");
02106 };
02107
02108 #if DEBUG_INTERFACE
02109     fprintf (stderr, "window_new: start\n");
02110 #endif
02111
02112 // Creating the window
02113 window->window = main_window
02114     = (GtkWindow *) gtk_application_window_new (application);
02115
02116 // Finish when closing the window
02117 g_signal_connect_swapped (window->window, "delete-event",
02118     G_CALLBACK (g_application_quit),
02119     G_APPLICATION (application));
02120
02121 // Setting the window title
02122 gtk_window_set_title (window->window, "MPCOTool");
02123
02124 // Creating the open button
02125 window->button_open = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("document-open",
02127         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02128 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02129
02130 // Creating the save button
02131 window->button_save = (GtkToolButton *) gtk_tool_button_new
02132     (gtk_image_new_from_icon_name ("document-save",
02133         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02134 g_signal_connect (window->button_save, "clicked", (GCallback)
02135     window_save,
02136     NULL);
02137
02138 // Creating the run button
02139 window->button_run = (GtkToolButton *) gtk_tool_button_new
02140     (gtk_image_new_from_icon_name ("system-run",
02141         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02142 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02143
02144 // Creating the options button
02145 window->button_options = (GtkToolButton *) gtk_tool_button_new
02146     (gtk_image_new_from_icon_name ("preferences-system",
02147         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02148 g_signal_connect (window->button_options, "clicked",
02149     options_new, NULL);
02150
02151 // Creating the help button
02152 window->button_help = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("help-browser",
02154         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02155 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02156
02157 // Creating the about button
02158 window->button_about = (GtkToolButton *) gtk_tool_button_new
02159     (gtk_image_new_from_icon_name ("help-about",
02160         GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02161 g_signal_connect (window->button_about, "clicked",
02162     window_about, NULL);
02163
02164 // Creating the exit button
02165 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02166     (gtk_image_new_from_icon_name ("application-exit",
02167         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02168 g_signal_connect_swapped (window->button_exit, "clicked",
02169     G_CALLBACK (g_application_quit),
02170     G_APPLICATION (application));
02171
02172 // Creating the buttons bar
02173 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02174 gtk_toolbar_insert
02175     (window->bar_buttons, GTK_TOOL_ITEM (window->
02176     button_open), 0);
02177 gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->
02179     button_save), 1);
02180 gtk_toolbar_insert
02181     (window->bar_buttons, GTK_TOOL_ITEM (window->
02182     button_run), 2);
02183 gtk_toolbar_insert
02184     (window->bar_buttons, GTK_TOOL_ITEM (window->
02185     button_options), 3);
02186 gtk_toolbar_insert
02187     (window->bar_buttons, GTK_TOOL_ITEM (window->
02188     button_help), 4);
02189 gtk_toolbar_insert
02190     (window->bar_buttons, GTK_TOOL_ITEM (window->

```

```

        button_about), 5);
02183     gtk_toolbar_insert
02184     (window->bar_buttons, GTK_TOOL_ITEM (window->
button_exit), 6);
02185     gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02186
02187     // Creating the simulator program label and entry
02188     window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02189     window->button_simulator = (GtkFileChooserButton *)
02190     gtk_file_chooser_button_new (_("Simulator program"),
02191     GTK_FILE_CHOOSER_ACTION_OPEN);
02192     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02193     _("Simulator program executable file"));
02194     gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02195
02196     // Creating the evaluator program label and entry
02197     window->check_evaluator = (GtkCheckButton *)
02198     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02199     g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
02200     window->button_evaluator = (GtkFileChooserButton *)
02201     gtk_file_chooser_button_new (_("Evaluator program"),
02202     GTK_FILE_CHOOSER_ACTION_OPEN);
02203     gtk_widget_set_tooltip_text
02204     (GTK_WIDGET (window->button_evaluator),
02205     _("Optional evaluator program executable file"));
02206
02207     // Creating the results files labels and entries
02208     window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02209     window->entry_result = (GtkEntry *) gtk_entry_new ();
02210     gtk_widget_set_tooltip_text
02211     (GTK_WIDGET (window->entry_result), _("Best results file"));
02212     window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02213     window->entry_variables = (GtkEntry *) gtk_entry_new ();
02214     gtk_widget_set_tooltip_text
02215     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02216
02217     // Creating the files grid and attaching widgets
02218     window->grid_files = (GtkGrid *) gtk_grid_new ();
02219     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02220     0, 0, 1, 1);
02221     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02222     1, 0, 1, 1);
02223     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02224     0, 1, 1, 1);
02225     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02226     1, 1, 1, 1);
02227     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02228     0, 2, 1, 1);
02229     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02230     1, 2, 1, 1);
02231     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02232     0, 3, 1, 1);
02233     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02234     1, 3, 1, 1);
02235
02236     // Creating the algorithm properties
02237     window->label_simulations = (GtkLabel *) gtk_label_new
02238     (_("Simulations number"));
02239     window->spin_simulations
02240     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02241     gtk_widget_set_tooltip_text
02242     (GTK_WIDGET (window->spin_simulations),
02243     _("Number of simulations to perform for each iteration"));
02244     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02245     window->label_iterations = (GtkLabel *)
02246     gtk_label_new (_("Iterations number"));
02247     window->spin_iterations
02248     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02249     gtk_widget_set_tooltip_text
02250     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02251     g_signal_connect
02252     (window->spin_iterations, "value-changed",
window_update, NULL);
02253     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02254     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02255     window->spin_tolerance =
02256     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02257     gtk_widget_set_tooltip_text

```

```

02258     (GTK_WIDGET (window->spin_tolerance),
02259      _("Tolerance to set the variable interval on the next iteration"));
02260 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02261 window->spin_bests
02262     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02263 gtk_widget_set_tooltip_text
02264     (GTK_WIDGET (window->spin_bests),
02265      _("Number of best simulations used to set the variable interval "
02266        "on the next iteration"));
02267 window->label_population
02268     = (GtkLabel *) gtk_label_new (_("Population number"));
02269 window->spin_population
02270     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02271 gtk_widget_set_tooltip_text
02272     (GTK_WIDGET (window->spin_population),
02273      _("Number of population for the genetic algorithm"));
02274 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02275 window->label_generations
02276     = (GtkLabel *) gtk_label_new (_("Generations number"));
02277 window->spin_generations
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02279 gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_generations),
02281      _("Number of generations for the genetic algorithm"));
02282 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02283 window->spin_mutation
02284     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02285 gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_mutation),
02287      _("Ratio of mutation for the genetic algorithm"));
02288 window->label_reproduction
02289     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02290 window->spin_reproduction
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292 gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_reproduction),
02294      _("Ratio of reproduction for the genetic algorithm"));
02295 window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02296 window->spin_adaptation
02297     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02298 gtk_widget_set_tooltip_text
02299     (GTK_WIDGET (window->spin_adaptation),
02300      _("Ratio of adaptation for the genetic algorithm"));
02301 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02302 window->spin_threshold = (GtkSpinButton *)
02303     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02304                                     precision[DEFAULT_PRECISION]);
02305 gtk_widget_set_tooltip_text
02306     (GTK_WIDGET (window->spin_threshold),
02307      _("Threshold in the objective function to finish the simulations"));
02308 window->scrolled_threshold =
02309     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02310 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02311                   GTK_WIDGET (window->spin_threshold));
02312 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02313 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02314 //                          GTK_ALIGN_FILL);
02315
02316 // Creating the direction search method properties
02317 window->check_direction = (GtkCheckButton *)
02318     gtk_check_button_new_with_mnemonic (_("Direction search method"));
02319 g_signal_connect (window->check_direction, "clicked",
02320                  window_update, NULL);
02321 window->grid_direction = (GtkGrid *) gtk_grid_new ();
02322 window->button_direction[0] = (GtkRadioButton *)
02323     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02324 gtk_grid_attach (window->grid_direction,
02325                 GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02326 g_signal_connect (window->button_direction[0], "clicked",
02327                  window_update, NULL);
02328 for (i = 0; ++i < NDIRECTIONS;)
02329 {
02330     window->button_direction[i] = (GtkRadioButton *)
02331         gtk_radio_button_new_with_mnemonic
02332             (gtk_radio_button_get_group (window->button_direction[0]),
02333              label_direction[i]);
02334     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02335                                 tip_direction[i]);
02336     gtk_grid_attach (window->grid_direction,
02337                     GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02338     g_signal_connect (window->button_direction[i], "clicked",
02339                      window_update, NULL);
02340 }
02341 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02342 window->spin_steps = (GtkSpinButton *)
02343     gtk_spin_button_new_with_range (1., 1.e12, 1.);

```

```

02343 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02344 window->label_estimates
02345 = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02346 window->spin_estimates = (GtkSpinButton *)
02347   gtk_spin_button_new_with_range (1., 1.e3, 1.);
02348 window->label_relaxation
02349 = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02350 window->spin_relaxation = (GtkSpinButton *)
02351   gtk_spin_button_new_with_range (0., 2., 0.001);
02352 gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02353   window->label_steps),
02354   0, NDIRECTIONS, 1, 1);
02355 gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02356   window->spin_steps),
02357   1, NDIRECTIONS, 1, 1);
02358 gtk_grid_attach (window->grid_direction,
02359   GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02360   1, 1);
02361 gtk_grid_attach (window->grid_direction,
02362   GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02363   1);
02364 gtk_grid_attach (window->grid_direction,
02365   GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02366   1, 1);
02367 gtk_grid_attach (window->grid_direction,
02368   GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02369   1, 1);
02370 // Creating the array of algorithms
02371 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02372 window->button_algorithm[0] = (GtkRadioButton *)
02373   gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02374 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02375   tip_algorithm[0]);
02376 gtk_grid_attach (window->grid_algorithm,
02377   GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02378 g_signal_connect (window->button_algorithm[0], "clicked",
02379   window_set_algorithm, NULL);
02380 for (i = 0; ++i < NALGORITHMS;)
02381 {
02382   window->button_algorithm[i] = (GtkRadioButton *)
02383     gtk_radio_button_new_with_mnemonic
02384       (gtk_radio_button_get_group (window->button_algorithm[0]),
02385        label_algorithm[i]);
02386   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02387     tip_algorithm[i]);
02388   gtk_grid_attach (window->grid_algorithm,
02389     GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02390   g_signal_connect (window->button_algorithm[i], "clicked",
02391     window_set_algorithm, NULL);
02392 }
02393 gtk_grid_attach (window->grid_algorithm,
02394   GTK_WIDGET (window->label_simulations), 0,
02395   NALGORITHMS, 1, 1);
02396 gtk_grid_attach (window->grid_algorithm,
02397   GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02398 gtk_grid_attach (window->grid_algorithm,
02399   GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02400   1, 1);
02401 gtk_grid_attach (window->grid_algorithm,
02402   GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02403   1, 1);
02404 gtk_grid_attach (window->grid_algorithm,
02405   GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02406   1, 1);
02407 gtk_grid_attach (window->grid_algorithm,
02408   GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02409   1);
02410 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02411   window->label_bests),
02412   0, NALGORITHMS + 3, 1, 1);
02413 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02414   window->spin_bests), 1,
02415   NALGORITHMS + 3, 1, 1);
02416 gtk_grid_attach (window->grid_algorithm,
02417   GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02418   1, 1);
02419 gtk_grid_attach (window->grid_algorithm,
02420   GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02421   1, 1);
02422 gtk_grid_attach (window->grid_algorithm,
02423   GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02424   1, 1);
02425 gtk_grid_attach (window->grid_algorithm,

```

```

02426         GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02427         1);
02428     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_mutation),
02429         1, NALGORITHMS + 6, 1, 1);
02430     gtk_grid_attach (window->grid_algorithm,
02431         GTK_WIDGET (window->label_reproduction), 0,
02432         NALGORITHMS + 7, 1, 1);
02433     gtk_grid_attach (window->grid_algorithm,
02434         GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02435         1, 1);
02436     gtk_grid_attach (window->grid_algorithm,
02437         GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02438         1, 1);
02439     gtk_grid_attach (window->grid_algorithm,
02440         GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02441         1, 1);
02442     gtk_grid_attach (window->grid_algorithm,
02443         GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02444         2, 1);
02445     gtk_grid_attach (window->grid_algorithm,
02446         GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02447         2, 1);
02448     gtk_grid_attach (window->grid_algorithm,
02449         GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02450         1, 1);
02451     gtk_grid_attach (window->grid_algorithm,
02452         GTK_WIDGET (window->scrolled_threshold), 1,
02453         NALGORITHMS + 11, 1, 1);
02454     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02455     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02456         GTK_WIDGET (window->grid_algorithm));
02457
02458     // Creating the variable widgets
02459     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02460     gtk_widget_set_tooltip_text
02461         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02462     window->id_variable = g_signal_connect
02463         (window->combo_variable, "changed", window_set_variable, NULL);
02464     window->button_add_variable
02465         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02466             GTK_ICON_SIZE_BUTTON);
02467     g_signal_connect
02468         (window->button_add_variable, "clicked",
window_add_variable, NULL);
02469     gtk_widget_set_tooltip_text
02470         (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02471     window->button_remove_variable
02472         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02473             GTK_ICON_SIZE_BUTTON);
02474     g_signal_connect
02475         (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
02476     gtk_widget_set_tooltip_text
02477         (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02478     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02479     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02480     gtk_widget_set_tooltip_text
02481         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02482     gtk_widget_set_expand (GTK_WIDGET (window->entry_variable), TRUE);
02483     window->id_variable_label = g_signal_connect
02484         (window->entry_variable, "changed",
window_label_variable, NULL);
02485     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02486     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02487         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02488     gtk_widget_set_tooltip_text
02489         (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02490     window->scrolled_min
02491         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02492     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02493         GTK_WIDGET (window->spin_min));
02494     g_signal_connect (window->spin_min, "value-changed",
02495         window_rangemin_variable, NULL);
02496     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02497     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02498         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02499     gtk_widget_set_tooltip_text
02500         (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02501     window->scrolled_max
02502         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02503     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02504         GTK_WIDGET (window->spin_max));
02505     g_signal_connect (window->spin_max, "value-changed",
02506         window_rangemax_variable, NULL);
02507     window->check_minabs = (GtkCheckButton *)
02508         gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));

```

```

02509 g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
02510 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02511 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02512 gtk_widget_set_tooltip_text
02513 (GTK_WIDGET (window->spin_minabs),
02514 _("Minimum allowed value of the variable"));
02515 window->scrolled_minabs
02516 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02517 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02518 GTK_WIDGET (window->spin_minabs));
02519 g_signal_connect (window->spin_minabs, "value-changed",
02520 window_rangeminabs_variable, NULL);
02521 window->check_maxabs = (GtkCheckButton *)
02522 gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02523 g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02524 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02525 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02526 gtk_widget_set_tooltip_text
02527 (GTK_WIDGET (window->spin_maxabs),
02528 _("Maximum allowed value of the variable"));
02529 window->scrolled_maxabs
02530 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02531 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02532 GTK_WIDGET (window->spin_maxabs));
02533 g_signal_connect (window->spin_maxabs, "value-changed",
02534 window_rangemaxabs_variable, NULL);
02535 window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02536 window->spin_precision = (GtkSpinButton *)
02537 gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02538 gtk_widget_set_tooltip_text
02539 (GTK_WIDGET (window->spin_precision),
02540 _("Number of precision floating point digits\n"
02541 "0 is for integer numbers"));
02542 g_signal_connect (window->spin_precision, "value-changed",
02543 window_precision_variable, NULL);
02544 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02545 window->spin_sweeps =
02546 (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02547 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02548 _("Number of steps sweeping the variable"));
02549 g_signal_connect (window->spin_sweeps, "value-changed",
02550 window_update_variable, NULL);
02551 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02552 window->spin_bits
02553 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02554 gtk_widget_set_tooltip_text
02555 (GTK_WIDGET (window->spin_bits),
02556 _("Number of bits to encode the variable"));
02557 g_signal_connect
02558 (window->spin_bits, "value-changed", window_update_variable, NULL)
;
02559 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02560 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02561 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02562 gtk_widget_set_tooltip_text
02563 (GTK_WIDGET (window->spin_step),
02564 _("Initial step size for the direction search method"));
02565 window->scrolled_step
02566 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02567 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02568 GTK_WIDGET (window->spin_step));
02569 g_signal_connect
02570 (window->spin_step, "value-changed", window_step_variable, NULL);
02571 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02572 gtk_grid_attach (window->grid_variable,
02573 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02574 gtk_grid_attach (window->grid_variable,
02575 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02576 gtk_grid_attach (window->grid_variable,
02577 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02578 gtk_grid_attach (window->grid_variable,
02579 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02580 gtk_grid_attach (window->grid_variable,
02581 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02582 gtk_grid_attach (window->grid_variable,
02583 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02584 gtk_grid_attach (window->grid_variable,
02585 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02586 gtk_grid_attach (window->grid_variable,
02587 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02588 gtk_grid_attach (window->grid_variable,
02589 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02590 gtk_grid_attach (window->grid_variable,
02591 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02592 gtk_grid_attach (window->grid_variable,

```

```

02593     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02594     gtk_grid_attach (window->grid_variable,
02595     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02596     gtk_grid_attach (window->grid_variable,
02597     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02598     gtk_grid_attach (window->grid_variable,
02599     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02600     gtk_grid_attach (window->grid_variable,
02601     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02602     gtk_grid_attach (window->grid_variable,
02603     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02604     gtk_grid_attach (window->grid_variable,
02605     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02606     gtk_grid_attach (window->grid_variable,
02607     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02608     gtk_grid_attach (window->grid_variable,
02609     GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02610     gtk_grid_attach (window->grid_variable,
02611     GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02612     gtk_grid_attach (window->grid_variable,
02613     GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02614     window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02615     gtk_container_add (GTK_CONTAINER (window->frame_variable),
02616     GTK_WIDGET (window->grid_variable));
02617
02618     // Creating the experiment widgets
02619     window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02620     gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02621     _("Experiment selector"));
02622     window->id_experiment = g_signal_connect
02623     (window->combo_experiment, "changed",
02624     window_set_experiment, NULL);
02625     window->button_add_experiment
02626     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02627     GTK_ICON_SIZE_BUTTON);
02628     g_signal_connect
02629     (window->button_add_experiment, "clicked",
02630     window_add_experiment, NULL);
02631     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02632     _("Add experiment"));
02633     window->button_remove_experiment
02634     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02635     GTK_ICON_SIZE_BUTTON);
02636     g_signal_connect (window->button_remove_experiment, "clicked",
02637     window_remove_experiment, NULL);
02638     gtk_widget_set_tooltip_text (GTK_WIDGET (window->
02639     button_remove_experiment),
02640     _("Remove experiment"));
02641     window->label_experiment
02642     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02643     window->button_experiment = (GtkFileChooserButton *)
02644     gtk_file_chooser_button_new (_("Experimental data file"),
02645     GTK_FILE_CHOOSER_ACTION_OPEN);
02646     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02647     _("Experimental data file"));
02648     window->id_experiment_name
02649     = g_signal_connect (window->button_experiment, "selection-changed",
02650     window_name_experiment, NULL);
02651     gtk_widget_set_hexpan (GTK_WIDGET (window->button_experiment), TRUE);
02652     window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02653     window->spin_weight
02654     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02655     gtk_widget_set_tooltip_text
02656     (GTK_WIDGET (window->spin_weight),
02657     _("Weight factor to build the objective function"));
02658     g_signal_connect
02659     (window->spin_weight, "value-changed",
02660     window_weight_experiment, NULL);
02661     window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02662     gtk_grid_attach (window->grid_experiment,
02663     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02664     gtk_grid_attach (window->grid_experiment,
02665     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02666     gtk_grid_attach (window->grid_experiment,
02667     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02668
02669     ;
02670     gtk_grid_attach (window->grid_experiment,
02671     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02672     gtk_grid_attach (window->grid_experiment,
02673     GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02674     gtk_grid_attach (window->grid_experiment,
02675     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02676     gtk_grid_attach (window->grid_experiment,
02677     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02678     for (i = 0; i < MAX_NINPITS; ++i)
02679     {
02680         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);

```



```

02675     window->check_template[i] = (GtkCheckButton *)
02676     gtk_check_button_new_with_label (buffer3);
02677     window->id_template[i]
02678     = g_signal_connect (window->check_template[i], "toggled",
02679     window_inputs_experiment, NULL);
02680     gtk_grid_attach (window->grid_experiment,
02681     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02682     window->button_template[i] =
02683     (GtkFileChooserButton *)
02684     gtk_file_chooser_button_new (_("Input template"),
02685     GTK_FILE_CHOOSER_ACTION_OPEN);
02686     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02687     _("Experimental input template file"));
02688     window->id_input[i] =
02689     g_signal_connect_swapped (window->button_template[i],
02690     "selection-changed",
02691     (GCallback) window_template_experiment,
02692     (void *) (size_t) i);
02693     gtk_grid_attach (window->grid_experiment,
02694     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02695 }
02696 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02697 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02698     GTK_WIDGET (window->grid_experiment));
02699
02700 // Creating the error norm widgets
02701 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02702 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02703 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02704     GTK_WIDGET (window->grid_norm));
02705 window->button_norm[0] = (GtkRadioButton *)
02706     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02707 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02708     tip_norm[0]);
02709 gtk_grid_attach (window->grid_norm,
02710     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02711 g_signal_connect (window->button_norm[0], "clicked",
02712     window_update, NULL);
02713 for (i = 0; ++i < NNORMS;)
02714 {
02715     window->button_norm[i] = (GtkRadioButton *)
02716     gtk_radio_button_new_with_mnemonic
02717     (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02718     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02719     tip_norm[i]);
02720     gtk_grid_attach (window->grid_norm,
02721     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02722     g_signal_connect (window->button_norm[i], "clicked",
02723     window_update, NULL);
02724 }
02725 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02726 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02727     label_p), 1, 1, 1, 1);
02728 window->spin_p =
02729     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02730     G_MAXDOUBLE, 0.01);
02731 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02732     _("P parameter for the P error norm"));
02733 window->scrolled_p =
02734     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02735 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02736     GTK_WIDGET (window->spin_p));
02737 gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02738 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02739 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02740     scrolled_p),
02741     1, 2, 1, 2);
02742
02743 // Creating the grid and attaching the widgets to the grid
02744 window->grid = (GtkGrid *) gtk_grid_new ();
02745 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02746 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02747 gtk_grid_attach (window->grid,
02748     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02749 gtk_grid_attach (window->grid,
02750     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02751 gtk_grid_attach (window->grid,
02752     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02753 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02754 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
02755     window->grid));
02756
02757 // Setting the window logo
02758 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02759 gtk_window_set_icon (window->window, window->logo);
02760
02761 // Showing the window

```



```

02757     gtk_widget_show_all (GTK_WIDGET (window->window));
02758
02759     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02760     #if GTK_MINOR_VERSION >= 16
02761         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02762         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02763         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02764         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02765         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02766         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02767         gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02768     #endif
02769
02770     // Reading initial example
02771     input_new ();
02772     buffer2 = g_get_current_dir ();
02773     buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02774     g_free (buffer2);
02775     window_read (buffer);
02776     g_free (buffer);
02777
02778     #if DEBUG_INTERFACE
02779         fprintf (stderr, "window_new: start\n");
02780     #endif
02781 }

```

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 1877 of file [interface.c](#).

```

01878 {
01879     unsigned int i;
01880     char *buffer;
01881     #if DEBUG_INTERFACE
01882         fprintf (stderr, "window_read: start\n");
01883     #endif
01884
01885     // Reading new input file
01886     input_free ();
01887     input->result = input->variables = NULL;
01888     if (!input_open (filename))
01889     {
01890         #if DEBUG_INTERFACE
01891             fprintf (stderr, "window_read: end\n");
01892         #endif
01893         return 0;
01894     }
01895
01896     // Setting GTK+ widgets data
01897     gtk_entry_set_text (window->entry_result, input->result);
01898     gtk_entry_set_text (window->entry_variables, input->
variables);
01899     buffer = g_build_filename (input->directory, input->
simulator, NULL);

```

```

01900     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01901                                   (window->button_simulator), buffer);
01902     g_free (buffer);
01903     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01904                                   (size_t) input->evaluator);
01905     if (input->evaluator)
01906     {
01907         buffer = g_build_filename (input->directory, input->
01908 evaluator, NULL);
01909         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01910                                       (window->button_evaluator), buffer);
01911         g_free (buffer);
01912     }
01913     gtk_toggle_button_set_active
01914 (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
01915 algorithm]), TRUE);
01916     switch (input->algorithm)
01917     {
01918         case ALGORITHM_MONTE_CARLO:
01919             gtk_spin_button_set_value (window->spin_simulations,
01920                                         (gdouble) input->nsimulations);
01921         case ALGORITHM_SWEEP:
01922             gtk_spin_button_set_value (window->spin_iterations,
01923                                         (gdouble) input->niterations);
01924             gtk_spin_button_set_value (window->spin_bests, (gdouble)
01925 input->nbest);
01926             gtk_spin_button_set_value (window->spin_tolerance,
01927                                         input->tolerance);
01928             gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01929                                           (window->check_direction),
01930                                           input->nsteps);
01931             if (input->nsteps)
01932             {
01933                 gtk_toggle_button_set_active
01934 (GTK_TOGGLE_BUTTON (window->button_direction
01935 [input->direction]), TRUE);
01936                 gtk_spin_button_set_value (window->spin_steps,
01937                                             (gdouble) input->nsteps);
01938                 gtk_spin_button_set_value (window->spin_relaxation,
01939                                             (gdouble) input->relaxation);
01940                 switch (input->direction)
01941                 {
01942                     case DIRECTION_METHOD_RANDOM:
01943                         gtk_spin_button_set_value (window->spin_estimates,
01944                                                     (gdouble) input->nestimates);
01945                 }
01946             }
01947             break;
01948             default:
01949                 gtk_spin_button_set_value (window->spin_population,
01950                                             (gdouble) input->nsimulations);
01951                 gtk_spin_button_set_value (window->spin_generations,
01952                                             (gdouble) input->niterations);
01953                 gtk_spin_button_set_value (window->spin_mutation, input->
01954 mutation_ratio);
01955                 gtk_spin_button_set_value (window->spin_reproduction,
01956                                             input->reproduction_ratio);
01957                 gtk_spin_button_set_value (window->spin_adaptation,
01958                                             input->adaptation_ratio);
01959             }
01960             gtk_toggle_button_set_active
01961 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01962             gtk_spin_button_set_value (window->spin_p, input->p);
01963             gtk_spin_button_set_value (window->spin_threshold, input->
01964 threshold);
01965             g_signal_handler_block (window->combo_experiment, window->
01966 id_experiment);
01967             g_signal_handler_block (window->button_experiment,
01968                                     window->id_experiment_name);
01969             gtk_combo_box_text_remove_all (window->combo_experiment);
01970             for (i = 0; i < input->nexperiments; ++i)
01971                 gtk_combo_box_text_append_text (window->combo_experiment,
01972                                                 input->experiment[i].name);
01973             g_signal_handler_unblock
01974 (window->button_experiment, window->
01975 id_experiment_name);
01976             g_signal_handler_unblock (window->combo_experiment,
01977 window->id_experiment);
01978             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01979             g_signal_handler_block (window->combo_variable, window->
01980 id_variable);
01981             g_signal_handler_block (window->entry_variable, window->
01982 id_variable_label);
01983             gtk_combo_box_text_remove_all (window->combo_variable);
01984             for (i = 0; i < input->nvariables; ++i)
01985                 gtk_combo_box_text_append_text (window->combo_variable,
01986                                                 input->variable[i].name);

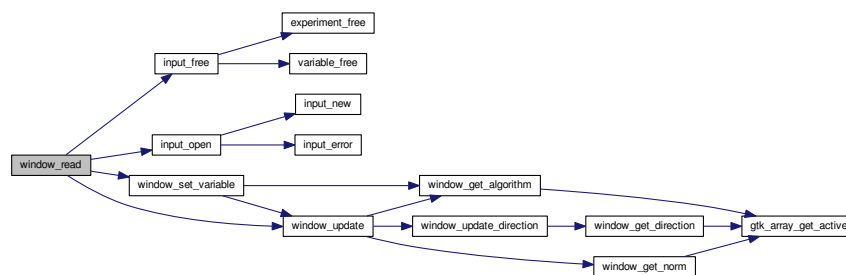
```

```

01975  g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01976  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01977  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01978  window_set_variable ();
01979  window_update ();
01980
01981  #if DEBUG_INTERFACE
01982  fprintf (stderr, "window_read: end\n");
01983  #endif
01984  return 1;
01985 }

```

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

```

00819 {
00820  GtkFileChooserDialog *dlg;
00821  GtkFileFilter *filter1, *filter2;
00822  char *buffer;
00823
00824  #if DEBUG_INTERFACE
00825  fprintf (stderr, "window_save: start\n");
00826  #endif
00827
00828  // Opening the saving dialog
00829  dlg = (GtkFileChooserDialog *)
00830  gtk_file_chooser_dialog_new (_, "Save file",
00831  window->window,
00832  GTK_FILE_CHOOSER_ACTION_SAVE,
00833  _("_Cancel"), GTK_RESPONSE_CANCEL,
00834  _("_OK"), GTK_RESPONSE_OK, NULL);
00835  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00836  buffer = g_build_filename (input->directory, input->name, NULL);
00837  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00838  g_free (buffer);
00839
00840  // Adding XML filter

```

```

00841 filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00842 gtk_file_filter_set_name (filter1, "XML");
00843 gtk_file_filter_add_pattern (filter1, "*.xml");
00844 gtk_file_filter_add_pattern (filter1, "*.XML");
00845 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00846
00847 // Adding JSON filter
00848 filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00849 gtk_file_filter_set_name (filter2, "JSON");
00850 gtk_file_filter_add_pattern (filter2, "*.json");
00851 gtk_file_filter_add_pattern (filter2, "*.JSON");
00852 gtk_file_filter_add_pattern (filter2, "*.js");
00853 gtk_file_filter_add_pattern (filter2, "*.JS");
00854 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00855
00856 if (input->type == INPUT_TYPE_XML)
00857     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00858 else
00859     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861 // If OK response then saving
00862 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00863 {
00864     // Setting input file type
00865     filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866     buffer = (char *) gtk_file_filter_get_name (filter1);
00867     if (!strcmp (buffer, "XML"))
00868         input->type = INPUT_TYPE_XML;
00869     else
00870         input->type = INPUT_TYPE_JSON;
00871
00872     // Adding properties to the root XML node
00873     input->simulator = gtk_file_chooser_get_filename
00874         (GTK_FILE_CHOOSER (window->button_simulator));
00875     if (gtk_toggle_button_get_active
00876         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00877         input->evaluator = gtk_file_chooser_get_filename
00878             (GTK_FILE_CHOOSER (window->button_evaluator));
00879     else
00880         input->evaluator = NULL;
00881     if (input->type == INPUT_TYPE_XML)
00882     {
00883         input->result
00884             = (char *) xmlStrdup ((const xmlChar *)
00885                 gtk_entry_get_text (window->entry_result));
00886         input->variables
00887             = (char *) xmlStrdup ((const xmlChar *)
00888                 gtk_entry_get_text (window->
00889                     entry_variables));
00890     }
00891     else
00892     {
00893         input->result = g_strdup (gtk_entry_get_text (window->
00894             entry_result));
00895         input->variables =
00896             g_strdup (gtk_entry_get_text (window->entry_variables));
00897     }
00898     // Setting the algorithm
00899     switch (window_get_algorithm ())
00900     {
00901     case ALGORITHM_MONTE_CARLO:
00902         input->algorithm = ALGORITHM_MONTE_CARLO;
00903         input->nsimulations
00904             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00905         input->niterations
00906             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00907         input->tolerance = gtk_spin_button_get_value (window->
00908             spin_tolerance);
00909         input->nbest = gtk_spin_button_get_value_as_int (window->
00910             spin_bests);
00911         window_save_direction ();
00912         break;
00913     case ALGORITHM_SWEEP:
00914         input->algorithm = ALGORITHM_SWEEP;
00915         input->niterations
00916             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917         input->tolerance = gtk_spin_button_get_value (window->
00918             spin_tolerance);
00919         input->nbest = gtk_spin_button_get_value_as_int (window->
00920             spin_bests);
00921         window_save_direction ();
00922         break;
00923     default:
00924         input->algorithm = ALGORITHM_GENETIC;
00925         input->nsimulations
00926             = gtk_spin_button_get_value_as_int (window->spin_population);

```

```

00922         input->niterations
00923         = gtk_spin_button_get_value_as_int (window->spin_generations);
00924         input->mutation_ratio
00925         = gtk_spin_button_get_value (window->spin_mutation);
00926         input->reproduction_ratio
00927         = gtk_spin_button_get_value (window->spin_reproduction);
00928         input->adaptation_ratio
00929         = gtk_spin_button_get_value (window->spin_adaptation);
00930         break;
00931     }
00932     input->norm = window_get_norm ();
00933     input->p = gtk_spin_button_get_value (window->spin_p);
00934     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00935
00936     // Saving the XML file
00937     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00938     input_save (buffer);
00939
00940     // Closing and freeing memory
00941     g_free (buffer);
00942     gtk_widget_destroy (GTK_WIDGET (dlg));
00943 #if DEBUG_INTERFACE
00944     fprintf (stderr, "window_save: end\n");
00945 #endif
00946     return 1;
00947 }
00948
00949 // Closing and freeing memory
00950 gtk_widget_destroy (GTK_WIDGET (dlg));
00951 #if DEBUG_INTERFACE
00952 fprintf (stderr, "window_save: end\n");
00953 #endif
00954 return 0;
00955 }

```

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 1521 of file [interface.c](#).

```

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

```

```

01543     fprintf (stderr, "window_template_experiment: end\n");
01544 #endif
01545 }

```

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 2012-2017, 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 *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *button_evaluator;
00074     GtkWidget *label_result;
00075     GtkWidget *entry_result;
00076     GtkWidget *label_variables;

```

```

00101   GtkWidget *entry_variables;
00102   GtkWidget *frame_norm;
00103   GtkWidget *grid_norm;
00104   GtkWidget *button_norm[NORMS];
00106   GtkWidget *label_p;
00107   GtkWidget *spin_p;
00108   GtkWidget *scrolled_p;
00110   GtkWidget *frame_algorithm;
00111   GtkWidget *grid_algorithm;
00112   GtkWidget *button_algorithm[NALGORITHMS];
00114   GtkWidget *label_simulations;
00115   GtkWidget *spin_simulations;
00117   GtkWidget *label_iterations;
00118   GtkWidget *spin_iterations;
00120   GtkWidget *label_tolerance;
00121   GtkWidget *spin_tolerance;
00122   GtkWidget *label_bests;
00123   GtkWidget *spin_bests;
00124   GtkWidget *label_population;
00125   GtkWidget *spin_population;
00127   GtkWidget *label_generations;
00128   GtkWidget *spin_generations;
00130   GtkWidget *label_mutation;
00131   GtkWidget *spin_mutation;
00132   GtkWidget *label_reproduction;
00133   GtkWidget *spin_reproduction;
00135   GtkWidget *label_adaptation;
00136   GtkWidget *spin_adaptation;
00138   GtkWidget *check_direction;
00140   GtkWidget *grid_direction;
00142   GtkWidget *button_direction[NDIRECTIONS];
00144   GtkWidget *label_steps;
00145   GtkWidget *spin_steps;
00146   GtkWidget *label_estimates;
00147   GtkWidget *spin_estimates;
00149   GtkWidget *label_relaxation;
00151   GtkWidget *spin_relaxation;
00153   GtkWidget *label_threshold;
00154   GtkWidget *spin_threshold;
00155   GtkWidget *scrolled_threshold;
00157   GtkWidget *frame_variable;
00158   GtkWidget *grid_variable;
00159   GtkWidget *comboBoxText *combo_variable;
00161   GtkWidget *button_add_variable;
00162   GtkWidget *button_remove_variable;
00163   GtkWidget *label_variable;
00164   GtkWidget *entry_variable;
00165   GtkWidget *label_min;
00166   GtkWidget *spin_min;
00167   GtkWidget *scrolled_min;
00168   GtkWidget *label_max;
00169   GtkWidget *spin_max;
00170   GtkWidget *scrolled_max;
00171   GtkWidget *check_minabs;
00172   GtkWidget *spin_minabs;
00173   GtkWidget *scrolled_minabs;
00174   GtkWidget *check_maxabs;
00175   GtkWidget *spin_maxabs;
00176   GtkWidget *scrolled_maxabs;
00177   GtkWidget *label_precision;
00178   GtkWidget *spin_precision;
00179   GtkWidget *label_sweeps;
00180   GtkWidget *spin_sweeps;
00181   GtkWidget *label_bits;
00182   GtkWidget *spin_bits;
00183   GtkWidget *label_step;
00184   GtkWidget *spin_step;
00185   GtkWidget *scrolled_step;
00186   GtkWidget *frame_experiment;
00187   GtkWidget *grid_experiment;
00188   GtkWidget *comboBoxText *combo_experiment;
00189   GtkWidget *button_add_experiment;
00190   GtkWidget *button_remove_experiment;
00191   GtkWidget *label_experiment;
00192   GtkWidget *FileChooserButton *button_experiment;
00194   GtkWidget *label_weight;
00195   GtkWidget *spin_weight;
00196   GtkWidget *check_template[MAX_NINPUTS];
00198   GtkWidget *FileChooserButton *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 n_experiments;
00213     unsigned int n_variables;
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.

```

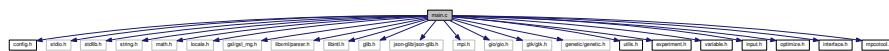
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>

```



```
#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"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
#include "mpcotool.h"
```

Include dependency graph for main.c:



Functions

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

4.15.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2017, all rights reserved.

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

4.16 main.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
```

```

00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #if HAVE_GTK
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
00053 #endif
00054 #include "genetic/genetic.h"
00055 #include "utils.h"
00056 #include "experiment.h"
00057 #include "variable.h"
00058 #include "input.h"
00059 #include "optimize.h"
00060 #if HAVE_GTK
00061 #include "interface.h"
00062 #endif
00063 #include "mpcotool.h"
00064
00065 int
00066 main (int argn, char **argc)
00067 {
00068     #if HAVE_GTK
00069         show_pending = process_pending;
00070     #endif
00071     return mpcotool (argn, argc);
00072 }

```

4.17 optimize.c File Reference

Source file to define the optimization functions.

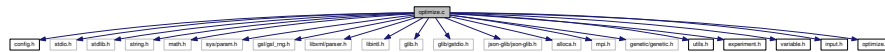
```

#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 *[stencil](#))
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

- unsigned int [nthreads_direction](#)
- *Number of threads for the direction search method.*
- 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 2012-2017, 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 [470](#) of file [optimize.c](#).

```
00471 {
00472     unsigned int i, j;
00473     double e;
00474     #if DEBUG_OPTIMIZE
00475     fprintf (stderr, "optimize_best: start\n");
00476     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00477             optimize->nsaveds, optimize->nbest);
00478     #endif
00479     if (optimize->nsaveds < optimize->nbest
00480         || value < optimize->error_best[optimize->nsaveds - 1])
00481     {
00482         if (optimize->nsaveds < optimize->nbest)
00483             ++optimize->nsaveds;
00484         optimize->error_best[optimize->nsaveds - 1] = value;
00485         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00486         for (i = optimize->nsaveds; --i;)
00487         {
00488             if (optimize->error_best[i] < optimize->
00489                 error_best[i - 1])
00489             {
00490                 j = optimize->simulation_best[i];
00491                 e = optimize->error_best[i];
00492                 optimize->simulation_best[i] = optimize->
00493                     simulation_best[i - 1];
```

```

00493         optimize->error_best[i] = optimize->
error_best[i - 1];
00494         optimize->simulation_best[i - 1] = j;
00495         optimize->error_best[i - 1] = e;
00496     }
00497     else
00498         break;
00499 }
00500 }
00501 #if DEBUG_OPTIMIZE
00502 fprintf (stderr, "optimize_best: end\n");
00503 #endif
00504 }

```

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

```

00798 {
00799 #if DEBUG_OPTIMIZE
00800     fprintf (stderr, "optimize_best_direction: start\n");
00801     fprintf (stderr,
00802         "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00803         simulation, value, optimize->error_best[0]);
00804 #endif
00805     if (value < optimize->error_best[0])
00806     {
00807         optimize->error_best[0] = value;
00808         optimize->simulation_best[0] = simulation;
00809 #if DEBUG_OPTIMIZE
00810         fprintf (stderr,
00811             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00812             simulation, value);
00813 #endif
00814     }
00815 #if DEBUG_OPTIMIZE
00816     fprintf (stderr, "optimize_best_direction: end\n");
00817 #endif
00818 }

```

4.17.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

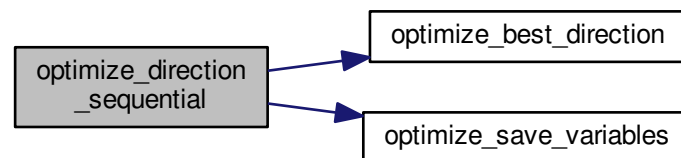
<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 827 of file [optimize.c](#).

```

00828 {
00829     unsigned int i, j;
00830     double e;
00831     #if DEBUG_OPTIMIZE
00832     fprintf (stderr, "optimize_direction_sequential: start\n");
00833     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00834             "nend_direction=%u\n",
00835             optimize->nstart_direction, optimize->
nend_direction);
00836     #endif
00837     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00838     {
00839         j = simulation + i;
00840         e = optimize_norm (j);
00841         optimize_best_direction (j, e);
00842         optimize_save_variables (j, e);
00843         if (e < optimize->threshold)
00844         {
00845             optimize->stop = 1;
00846             break;
00847         }
00848     #if DEBUG_OPTIMIZE
00849     fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00850     #endif
00851     }
00852     #if DEBUG_OPTIMIZE
00853     fprintf (stderr, "optimize_direction_sequential: end\n");
00854     #endif
00855 }
```

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

```

00866 {
00867     unsigned int i, thread;
00868     double e;
00869     #if DEBUG_OPTIMIZE
00870     fprintf (stderr, "optimize_direction_thread: start\n");
00871     #endif
00872     thread = data->thread;
00873     #if DEBUG_OPTIMIZE
00874     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00875             thread,
00876             optimize->thread_direction[thread],
00877             optimize->thread_direction[thread + 1]);
00878     #endif
00879     for (i = optimize->thread_direction[thread];
00880          i < optimize->thread_direction[thread + 1]; ++i)
00881     {
00882         e = optimize_norm (i);
00883         g_mutex_lock (mutex);
00884         optimize_best_direction (i, e);
00885         optimize_save_variables (i, e);
00886         if (e < optimize->threshold)
00887             optimize->stop = 1;
00888         g_mutex_unlock (mutex);
00889         if (optimize->stop)
00890             break;
00891     #if DEBUG_OPTIMIZE
00892     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00893     #endif
00894     }
00895     #if DEBUG_OPTIMIZE
00896     fprintf (stderr, "optimize_direction_thread: end\n");
00897     #endif
00898     g_thread_exit (NULL);
00899     return NULL;
00900 }

```

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


```

00941 {
00942     double x;
00943     #if DEBUG_OPTIMIZE
00944     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00945     #endif
00946     x = optimize->direction[variable];
00947     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00948     {
00949         if (estimate & 1)
00950             x += optimize->step[variable];
00951         else
00952             x -= optimize->step[variable];
00953     }
00954     #if DEBUG_OPTIMIZE
00955     fprintf (stderr,
00956             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00957             variable, x);
00958     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00959     #endif
00960     return x;
00961 }

```

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

```

00914 {
00915     double x;
00916     #if DEBUG_OPTIMIZE
00917     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00918     #endif
00919     x = optimize->direction[variable]
00920         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00921         step[variable];
00922     #if DEBUG_OPTIMIZE
00923     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00924             variable, x);
00925     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00926     #endif
00927     return x;
00928 }

```

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

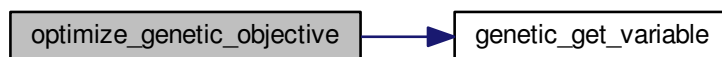
objective function value.

Definition at line 1106 of file [optimize.c](#).

```

01107 {
01108     unsigned int j;
01109     double objective;
01110     char buffer[64];
01111     #if DEBUG_OPTIMIZE
01112     fprintf (stderr, "optimize_genetic_objective: start\n");
01113     #endif
01114     for (j = 0; j < optimize->nvariables; ++j)
01115     {
01116         optimize->value[entity->id * optimize->nvariables + j]
01117             = genetic_get_variable (entity, optimize->genetic_variable + j);
01118     }
01119     objective = optimize_norm (entity->id);
01120     g_mutex_lock (mutex);
01121     for (j = 0; j < optimize->nvariables; ++j)
01122     {
01123         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01124         fprintf (optimize->file_variables, buffer,
01125             genetic_get_variable (entity, optimize->genetic_variable + j));
01126     }
01127     fprintf (optimize->file_variables, "%.14le\n", objective);
01128     g_mutex_unlock (mutex);
01129     #if DEBUG_OPTIMIZE
01130     fprintf (stderr, "optimize_genetic_objective: end\n");
01131     #endif
01132     return objective;
01133 }
```

Here is the call graph for this function:



4.17.2.8 optimize_input()

```

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

Function to write the simulation input file.

Parameters

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

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

```

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

```

```

00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177     return;
00178 }

```

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

```

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

```

```

00638     optimize->nsaveds = k;
00639     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00640     memcpy (optimize->error_best, e, k * sizeof (double));
00641     #if DEBUG_OPTIMIZE
00642     fprintf (stderr, "optimize_merge: end\n");
00643     #endif
00644 }

```

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.

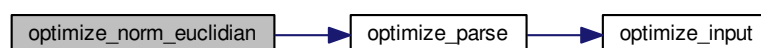
Definition at line [302](#) of file [optimize.c](#).

```

00303 {
00304     double e, ei;
00305     unsigned int i;
00306     #if DEBUG_OPTIMIZE
00307     fprintf (stderr, "optimize_norm_euclidian: start\n");
00308     #endif
00309     e = 0.;
00310     for (i = 0; i < optimize->nexperiments; ++i)
00311     {
00312         ei = optimize_parse (simulation, i);
00313         e += ei * ei;
00314     }
00315     e = sqrt (e);
00316     #if DEBUG_OPTIMIZE
00317     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00318     fprintf (stderr, "optimize_norm_euclidian: end\n");
00319     #endif
00320     return e;
00321 }

```

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.
-------------------	--------------------

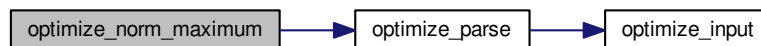
Returns

Maximum error norm.

Definition at line 331 of file [optimize.c](#).

```
00332 {
00333     double e, ei;
00334     unsigned int i;
00335     #if DEBUG_OPTIMIZE
00336     fprintf (stderr, "optimize_norm_maximum: start\n");
00337     #endif
00338     e = 0.;
00339     for (i = 0; i < optimize->nexperiments; ++i)
00340     {
00341         ei = fabs (optimize_parse (simulation, i));
00342         e = fmax (e, ei);
00343     }
00344     #if DEBUG_OPTIMIZE
00345     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00346     fprintf (stderr, "optimize_norm_maximum: end\n");
00347     #endif
00348     return e;
00349 }
```

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.
-------------------	--------------------

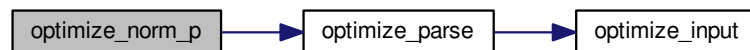
Returns

P error norm.

Definition at line 359 of file [optimize.c](#).

```
00360 {
00361     double e, ei;
00362     unsigned int i;
00363     #if DEBUG_OPTIMIZE
00364     fprintf (stderr, "optimize_norm_p: start\n");
00365     #endif
00366     e = 0.;
00367     for (i = 0; i < optimize->nexperiments; ++i)
00368     {
00369         ei = fabs (optimize_parse (simulation, i));
00370         e += pow (ei, optimize->p);
00371     }
00372     e = pow (e, 1. / optimize->p);
00373     #if DEBUG_OPTIMIZE
00374     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00375     fprintf (stderr, "optimize_norm_p: end\n");
00376     #endif
00377     return e;
00378 }
```

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.
-------------------	--------------------

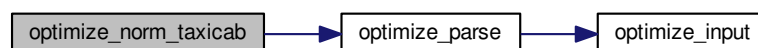
Returns

Taxicab error norm.

Definition at line 388 of file [optimize.c](#).

```
00389 {
00390     double e;
00391     unsigned int i;
00392     #if DEBUG_OPTIMIZE
00393     fprintf (stderr, "optimize_norm_taxicab: start\n");
00394     #endif
00395     e = 0.;
00396     for (i = 0; i < optimize->nexperiments; ++i)
00397         e += fabs (optimize_parse (simulation, i));
00398     #if DEBUG_OPTIMIZE
00399     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00400     fprintf (stderr, "optimize_norm_taxicab: end\n");
00401     #endif
00402     return e;
00403 }
```

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, experiment);
00209     #if DEBUG_OPTIMIZE
00210         fprintf(stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00211     #endif
00212         optimize_input (simulation, &input[i][0], optimize->
00213             file[i][experiment]);
00214     }
00215     for (; i < MAX_NINPUTS; ++i)
00216         strcpy (&input[i][0], "");
00217     #if DEBUG_OPTIMIZE
00218     fprintf(stderr, "optimize_parse: parsing end\n");
00219     #endif
00220
00221     // Performing the simulation
00222     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00223     buffer2 = g_path_get_dirname (optimize->simulator);
00224     buffer3 = g_path_get_basename (optimize->simulator);
00225     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00226     snprintf (buffer, 512, "%s\n" %s %s %s %s %s %s %s %s %s",
00227             buffer4, input[0], input[1], input[2], input[3], input[4],
00228             input[5], input[6], input[7], output);
00229     g_free (buffer4);
00230     g_free (buffer3);
00231     g_free (buffer2);
00232     #if DEBUG_OPTIMIZE
00233     fprintf(stderr, "optimize_parse: %s\n", buffer);
00234     #endif
00235     system (buffer);
00236
00237     // Checking the objective value function
00238     if (optimize->evaluator)
00239     {
00240         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00241         buffer2 = g_path_get_dirname (optimize->evaluator);
00242         buffer3 = g_path_get_basename (optimize->evaluator);
00243         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00244         snprintf (buffer, 512, "%s\n" %s %s %s",
00245             buffer4, output, optimize->experiment[experiment], result);
00246         g_free (buffer4);
00247         g_free (buffer3);
00248         g_free (buffer2);
00249     #if DEBUG_OPTIMIZE
00250         fprintf(stderr, "optimize_parse: %s\n", buffer);
00251         fprintf(stderr, "optimize_parse: result=%s\n", result);
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     #if DEBUG_OPTIMIZE
00261         fprintf(stderr, "optimize_parse: output=%s\n", output);
00262     #endif
00263         strcpy (result, "");
00264         file_result = g_fopen (output, "r");
00265         e = atof (fgets (buffer, 512, file_result));
00266         fclose (file_result);
00267     }
00268
00269     // Removing files
00270     #if !DEBUG_OPTIMIZE
00271     for (i = 0; i < optimize->ninputs; ++i)
00272     {
00273         if (optimize->file[i][0])
00274         {
00275             snprintf (buffer, 512, RM " %s", &input[i][0]);
00276             system (buffer);
00277         }
00278     }
00279     snprintf (buffer, 512, RM " %s %s", output, result);
00280     system (buffer);
00281     #endif

```

```

00281
00282 // Processing pending events
00283 if (show_pending)
00284     show_pending ();
00285
00286 #if DEBUG_OPTIMIZE
00287     fprintf (stderr, "optimize_parse: end\n");
00288 #endif
00289
00290 // Returning the objective function
00291 return e * optimize->weight[experiment];
00292 }

```

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

```

00442 {
00443     unsigned int i;
00444     char buffer[64];
00445     #if DEBUG_OPTIMIZE
00446     fprintf (stderr, "optimize_save_variables: start\n");
00447     #endif
00448     for (i = 0; i < optimize->nvariables; ++i)
00449     {
00450         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00451         fprintf (optimize->file_variables, buffer,
00452             optimize->value[simulation * optimize->
00453                 nvariables + i]);
00454     }
00454     fprintf (optimize->file_variables, "%.14le\n", error);
00455     fflush (optimize->file_variables);
00456     #if DEBUG_OPTIMIZE
00457     fprintf (stderr, "optimize_save_variables: end\n");
00458     #endif
00459 }

```

4.17.2.16 optimize_step_direction()

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

Function to do a step of the direction search method.

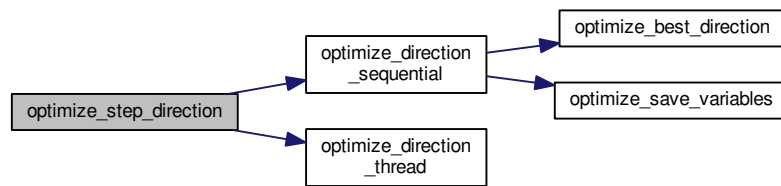
Parameters

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

Definition at line 970 of file [optimize.c](#).

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

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

```

00548 {
00549     unsigned int i, thread;
00550     double e;
00551     #if DEBUG_OPTIMIZE
00552         fprintf (stderr, "optimize_thread: start\n");
00553     #endif
00554     thread = data->thread;
00555     #if DEBUG_OPTIMIZE
00556         fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00557                 optimize->thread[thread], optimize->thread[thread + 1]);
00558     #endif
00559     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00560     {
00561         e = optimize_norm (i);
00562         g_mutex_lock (mutex);
00563         optimize_best (i, e);
00564         optimize_save_variables (i, e);
00565         if (e < optimize->threshold)
00566             optimize->stop = 1;
00567         g_mutex_unlock (mutex);
00568         if (optimize->stop)
00569             break;
00570     #if DEBUG_OPTIMIZE
00571         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00572     #endif
00573     }
00574     #if DEBUG_OPTIMIZE
00575         fprintf (stderr, "optimize_thread: end\n");
00576     #endif
00577     g_thread_exit (NULL);
00578     return NULL;
00579 }
```

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 Burguete and Borja Latorre.
00007
00008 Copyright 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <sys/param.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <glib/gstdio.h>
00044 #include <json-glib/json-glib.h>
00045 #ifdef G_OS_WIN32
00046 #include <windows.h>
00047 #elif !defined(__BSD_VISIBLE) && !defined(NetBSD)
00048 #include <alloca.h>
00049 #endif
00050 #if HAVE_MPI
00051 #include <mpi.h>
00052 #endif
00053 #include "genetic/genetic.h"
00054 #include "utils.h"
00055 #include "experiment.h"
00056 #include "variable.h"
00057 #include "input.h"
00058 #include "optimize.h"
00059
00060 #define DEBUG_OPTIMIZE 0
00061
00062 #ifdef G_OS_WIN32
00063 #define RM "del"
00064 #else
00065 #define RM "rm"
00066 #endif
00067
00068 unsigned int nthreads_direction;
00069 void (*optimize_algorithm) ();
00070 double (*optimize_estimate_direction) (unsigned int variable,
00071                                       unsigned int estimate);
00072 double (*optimize_norm) (unsigned int simulation);
00073 Optimize optimize[1];
00074
00075 void
00076 optimize_input (unsigned int simulation, char *input, GMappedFile *
00077 stencil)
00078 {
00079     unsigned int i;
00080     char buffer[32], value[32], *buffer2, *buffer3, *content;
00081     FILE *file;
00082     gsize length;
00083     GRegex *regex;

```

```

00108
00109 #if DEBUG_OPTIMIZE
00110     fprintf (stderr, "optimize_input: start\n");
00111 #endif
00112
00113     // Checking the file
00114     if (!stencil)
00115         goto optimize_input_end;
00116
00117     // Opening stencil
00118     content = g_mapped_file_get_contents (stencil);
00119     length = g_mapped_file_get_length (stencil);
00120 #if DEBUG_OPTIMIZE
00121     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00122 #endif
00123     file = g_fopen (input, "w");
00124
00125     // Parsing stencil
00126     for (i = 0; i < optimize->nvariables; ++i)
00127     {
00128 #if DEBUG_OPTIMIZE
00129         fprintf (stderr, "optimize_input: variable=%u\n", i);
00130 #endif
00131         snprintf (buffer, 32, "@variable%u@", i + 1);
00132         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00133                             NULL);
00134         if (i == 0)
00135         {
00136             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00137                                                optimize->label[i],
00138                                                (GRegexMatchFlags) 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],
00148                                                (GRegexMatchFlags) 0, NULL);
00149             g_free (buffer3);
00150         }
00151         g_regex_unref (regex);
00152         length = strlen (buffer2);
00153         snprintf (buffer, 32, "@value%u@", i + 1);
00154         regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00155                             NULL);
00156         snprintf (value, 32, format[optimize->precision[i]],
00157                  optimize->value[simulation * optimize->nvariables + i]);
00158 #if DEBUG_OPTIMIZE
00159         fprintf (stderr, "optimize_input: value=%s\n", value);
00160 #endif
00161         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00162                                           (GRegexMatchFlags) 0, NULL);
00163         g_free (buffer2);
00164         g_regex_unref (regex);
00165     }
00166
00167     // Saving input file
00168     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00169     g_free (buffer3);
00170     fclose (file);
00171
00172 optimize_input_end:
00173 #if DEBUG_OPTIMIZE
00174     fprintf (stderr, "optimize_input: end\n");
00175 #endif
00176     return;
00177 }
00178
00179 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, experiment);
00209 #if DEBUG_OPTIMIZE
00210     fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00211 #endif
00212     optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00213 }
00214 for (; i < MAX_NINPUTS; ++i)
00215     strcpy (&input[i][0], "");
00216 #if DEBUG_OPTIMIZE
00217 fprintf (stderr, "optimize_parse: parsing end\n");
00218 #endif
00219
00220 // Performing the simulation
00221 snprintf (output, 32, "output-%u-%u", simulation, experiment);
00222 buffer2 = g_path_get_dirname (optimize->simulator);
00223 buffer3 = g_path_get_basename (optimize->simulator);
00224 buffer4 = g_build_filename (buffer2, buffer3, NULL);
00225 snprintf (buffer, 512, "%s\\%s %s %s %s %s %s %s %s %s",
00226     buffer4, input[0], input[1], input[2], input[3], input[4],
00227     input[5], input[6], input[7], output);
00228 g_free (buffer4);
00229 g_free (buffer3);
00230 g_free (buffer2);
00231 #if DEBUG_OPTIMIZE
00232 fprintf (stderr, "optimize_parse: %s\n", buffer);
00233 #endif
00234 system (buffer);
00235
00236 // Checking the objective value function
00237 if (optimize->evaluator)
00238 {
00239     snprintf (result, 32, "result-%u-%u", simulation, experiment);
00240     buffer2 = g_path_get_dirname (optimize->evaluator);
00241     buffer3 = g_path_get_basename (optimize->evaluator);
00242     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00243     snprintf (buffer, 512, "%s\\%s %s %s %s",
00244         buffer4, output, optimize->experiment[experiment], result);
00245     g_free (buffer4);
00246     g_free (buffer3);
00247     g_free (buffer2);
00248 #if DEBUG_OPTIMIZE
00249     fprintf (stderr, "optimize_parse: %s\n", buffer);
00250     fprintf (stderr, "optimize_parse: result=%s\n", result);
00251 #endif
00252     system (buffer);
00253     file_result = g_fopen (result, "r");
00254     e = atof (fgets (buffer, 512, file_result));
00255     fclose (file_result);
00256 }
00257 else
00258 {
00259 #if DEBUG_OPTIMIZE
00260     fprintf (stderr, "optimize_parse: output=%s\n", output);
00261 #endif
00262     strcpy (result, "");
00263     file_result = g_fopen (output, "r");
00264     e = atof (fgets (buffer, 512, file_result));
00265     fclose (file_result);
00266 }
00267
00268 // Removing files
00269 #if !DEBUG_OPTIMIZE
00270 for (i = 0; i < optimize->ninputs; ++i)
00271 {
00272     if (optimize->file[i][0])
00273     {
00274         snprintf (buffer, 512, RM " %s", &input[i][0]);
00275         system (buffer);
00276     }
00277 }
00278     snprintf (buffer, 512, RM " %s %s", output, result);
00279     system (buffer);
00280 #endif
00281
00282 // Processing pending events
00283 if (show_pending)
00284     show_pending ();
00285
00286 #if DEBUG_OPTIMIZE
00287 fprintf (stderr, "optimize_parse: end\n");
00288 #endif
00289
00290 // Returning the objective function
00291 return e * optimize->weight[experiment];

```

```

00292 }
00293
00301 double
00302 optimize_norm_euclidian (unsigned int simulation)
00303 {
00304     double e, ei;
00305     unsigned int i;
00306     #if DEBUG_OPTIMIZE
00307     fprintf (stderr, "optimize_norm_euclidian: start\n");
00308     #endif
00309     e = 0.;
00310     for (i = 0; i < optimize->nexperiments; ++i)
00311     {
00312         ei = optimize_parse (simulation, i);
00313         e += ei * ei;
00314     }
00315     e = sqrt (e);
00316     #if DEBUG_OPTIMIZE
00317     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00318     fprintf (stderr, "optimize_norm_euclidian: end\n");
00319     #endif
00320     return e;
00321 }
00322
00330 double
00331 optimize_norm_maximum (unsigned int simulation)
00332 {
00333     double e, ei;
00334     unsigned int i;
00335     #if DEBUG_OPTIMIZE
00336     fprintf (stderr, "optimize_norm_maximum: start\n");
00337     #endif
00338     e = 0.;
00339     for (i = 0; i < optimize->nexperiments; ++i)
00340     {
00341         ei = fabs (optimize_parse (simulation, i));
00342         e = fmax (e, ei);
00343     }
00344     #if DEBUG_OPTIMIZE
00345     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00346     fprintf (stderr, "optimize_norm_maximum: end\n");
00347     #endif
00348     return e;
00349 }
00350
00358 double
00359 optimize_norm_p (unsigned int simulation)
00360 {
00361     double e, ei;
00362     unsigned int i;
00363     #if DEBUG_OPTIMIZE
00364     fprintf (stderr, "optimize_norm_p: start\n");
00365     #endif
00366     e = 0.;
00367     for (i = 0; i < optimize->nexperiments; ++i)
00368     {
00369         ei = fabs (optimize_parse (simulation, i));
00370         e += pow (ei, optimize->p);
00371     }
00372     e = pow (e, 1. / optimize->p);
00373     #if DEBUG_OPTIMIZE
00374     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00375     fprintf (stderr, "optimize_norm_p: end\n");
00376     #endif
00377     return e;
00378 }
00379
00387 double
00388 optimize_norm_taxicab (unsigned int simulation)
00389 {
00390     double e;
00391     unsigned int i;
00392     #if DEBUG_OPTIMIZE
00393     fprintf (stderr, "optimize_norm_taxicab: start\n");
00394     #endif
00395     e = 0.;
00396     for (i = 0; i < optimize->nexperiments; ++i)
00397         e += fabs (optimize_parse (simulation, i));
00398     #if DEBUG_OPTIMIZE
00399     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00400     fprintf (stderr, "optimize_norm_taxicab: end\n");
00401     #endif
00402     return e;
00403 }
00404
00409 void
00410 optimize_print ()

```



```

00411 {
00412     unsigned int i;
00413     char buffer[512];
00414     #if HAVE_MPI
00415     if (optimize->mpi_rank)
00416         return;
00417     #endif
00418     printf ("%s\n", _("Best result"));
00419     fprintf (optimize->file_result, "%s\n", _("Best result"));
00420     printf ("error = %.15le\n", optimize->error_old[0]);
00421     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00422     for (i = 0; i < optimize->nvariables; ++i)
00423     {
00424         snprintf (buffer, 512, "%s = %s\n",
00425                 optimize->label[i], format[optimize->precision[i]]);
00426         printf (buffer, optimize->value_old[i]);
00427         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00428     }
00429     fflush (optimize->file_result);
00430 }
00431
00440 void
00441 optimize_save_variables (unsigned int simulation, double error)
00442 {
00443     unsigned int i;
00444     char buffer[64];
00445     #if DEBUG_OPTIMIZE
00446     fprintf (stderr, "optimize_save_variables: start\n");
00447     #endif
00448     for (i = 0; i < optimize->nvariables; ++i)
00449     {
00450         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00451         fprintf (optimize->file_variables, buffer,
00452                 optimize->value[simulation * optimize->nvariables + i]);
00453     }
00454     fprintf (optimize->file_variables, "%.14le\n", error);
00455     fflush (optimize->file_variables);
00456     #if DEBUG_OPTIMIZE
00457     fprintf (stderr, "optimize_save_variables: end\n");
00458     #endif
00459 }
00460
00469 void
00470 optimize_best (unsigned int simulation, double value)
00471 {
00472     unsigned int i, j;
00473     double e;
00474     #if DEBUG_OPTIMIZE
00475     fprintf (stderr, "optimize_best: start\n");
00476     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00477             optimize->nsaveds, optimize->nbest);
00478     #endif
00479     if (optimize->nsaveds < optimize->nbest
00480         || value < optimize->error_best[optimize->nsaveds - 1])
00481     {
00482         if (optimize->nsaveds < optimize->nbest)
00483             ++optimize->nsaveds;
00484         optimize->error_best[optimize->nsaveds - 1] = value;
00485         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00486         for (i = optimize->nsaveds; --i;)
00487         {
00488             if (optimize->error_best[i] < optimize->error_best[i - 1])
00489             {
00490                 j = optimize->simulation_best[i];
00491                 e = optimize->error_best[i];
00492                 optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00493                 optimize->error_best[i] = optimize->error_best[i - 1];
00494                 optimize->simulation_best[i - 1] = j;
00495                 optimize->error_best[i - 1] = e;
00496             }
00497             else
00498                 break;
00499         }
00500     }
00501     #if DEBUG_OPTIMIZE
00502     fprintf (stderr, "optimize_best: end\n");
00503     #endif
00504 }
00505
00510 void
00511 optimize_sequential ()
00512 {
00513     unsigned int i;
00514     double e;
00515     #if DEBUG_OPTIMIZE

```

```

00516     fprintf (stderr, "optimize_sequential: start\n");
00517     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00518             optimize->nstart, optimize->nend);
00519 #endif
00520     for (i = optimize->nstart; i < optimize->nend; ++i)
00521     {
00522         e = optimize_norm (i);
00523         optimize_best (i, e);
00524         optimize_save_variables (i, e);
00525         if (e < optimize->threshold)
00526         {
00527             optimize->stop = 1;
00528             break;
00529         }
00530 #if DEBUG_OPTIMIZE
00531         fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00532 #endif
00533     }
00534 #if DEBUG_OPTIMIZE
00535     fprintf (stderr, "optimize_sequential: end\n");
00536 #endif
00537 }
00538
00546 void *
00547 optimize_thread (ParallelData * data)
00548 {
00549     unsigned int i, thread;
00550     double e;
00551 #if DEBUG_OPTIMIZE
00552     fprintf (stderr, "optimize_thread: start\n");
00553 #endif
00554     thread = data->thread;
00555 #if DEBUG_OPTIMIZE
00556     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00557             optimize->thread[thread], optimize->thread[thread + 1]);
00558 #endif
00559     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00560     {
00561         e = optimize_norm (i);
00562         g_mutex_lock (mutex);
00563         optimize_best (i, e);
00564         optimize_save_variables (i, e);
00565         if (e < optimize->threshold)
00566         {
00567             optimize->stop = 1;
00568             g_mutex_unlock (mutex);
00569             if (optimize->stop)
00570                 break;
00571 #if DEBUG_OPTIMIZE
00572             fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00573 #endif
00574 #if DEBUG_OPTIMIZE
00575             fprintf (stderr, "optimize_thread: end\n");
00576 #endif
00577             g_thread_exit (NULL);
00578             return NULL;
00579         }
00580     }
00592 void
00593 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00594                double *error_best)
00595 {
00596     unsigned int i, j, k, s[optimize->nbest];
00597     double e[optimize->nbest];
00598 #if DEBUG_OPTIMIZE
00599     fprintf (stderr, "optimize_merge: start\n");
00600 #endif
00601     i = j = k = 0;
00602     do
00603     {
00604         if (i == optimize->nsaveds)
00605         {
00606             s[k] = simulation_best[j];
00607             e[k] = error_best[j];
00608             ++j;
00609             ++k;
00610             if (j == nsaveds)
00611                 break;
00612         }
00613         else if (j == nsaveds)
00614         {
00615             s[k] = optimize->simulation_best[i];
00616             e[k] = optimize->error_best[i];
00617             ++i;
00618             ++k;
00619             if (i == optimize->nsaveds)
00620                 break;

```

```

00621     }
00622     else if (optimize->error_best[i] > error_best[j])
00623     {
00624         s[k] = simulation_best[j];
00625         e[k] = error_best[j];
00626         ++j;
00627         ++k;
00628     }
00629     else
00630     {
00631         s[k] = optimize->simulation_best[i];
00632         e[k] = optimize->error_best[i];
00633         ++i;
00634         ++k;
00635     }
00636 }
00637 while (k < optimize->nbest);
00638 optimize->nsaveds = k;
00639 memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00640 memcpy (optimize->error_best, e, k * sizeof (double));
00641 #if DEBUG_OPTIMIZE
00642 fprintf (stderr, "optimize_merge: end\n");
00643 #endif
00644 }
00645
00650 #if HAVE_MPI
00651 void
00652 optimize_synchronise ()
00653 {
00654     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00655     double error_best[optimize->nbest];
00656     MPI_Status mpi_stat;
00657     #if DEBUG_OPTIMIZE
00658     fprintf (stderr, "optimize_synchronise: start\n");
00659     #endif
00660     if (optimize->mpi_rank == 0)
00661     {
00662         for (i = 1; i < ntasks; ++i)
00663         {
00664             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00665             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00666                     MPI_COMM_WORLD, &mpi_stat);
00667             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00668                     MPI_COMM_WORLD, &mpi_stat);
00669             optimize_merge (nsaveds, simulation_best, error_best);
00670             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00671             if (stop)
00672                 optimize->stop = 1;
00673         }
00674         for (i = 1; i < ntasks; ++i)
00675             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00676     }
00677     else
00678     {
00679         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00680         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00681                 MPI_COMM_WORLD);
00682         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00683                 MPI_COMM_WORLD);
00684         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00685         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00686         if (stop)
00687             optimize->stop = 1;
00688     }
00689     #if DEBUG_OPTIMIZE
00690     fprintf (stderr, "optimize_synchronise: end\n");
00691     #endif
00692 }
00693 #endif
00694
00699 void
00700 optimize_sweep ()
00701 {
00702     unsigned int i, j, k, l;
00703     double e;
00704     GThread *thread[nthreads];
00705     ParallelData data[nthreads];
00706     #if DEBUG_OPTIMIZE
00707     fprintf (stderr, "optimize_sweep: start\n");
00708     #endif
00709     for (i = 0; i < optimize->nsimulations; ++i)
00710     {
00711         k = i;
00712         for (j = 0; j < optimize->nvariables; ++j)
00713         {
00714             l = k % optimize->nsweeps[j];
00715             k /= optimize->nsweeps[j];

```

```

00716         e = optimize->rangemin[j];
00717         if (optimize->nsweeps[j] > 1)
00718             e += 1 * (optimize->rangemax[j] - optimize->rangemin[j])
00719                 / (optimize->nsweeps[j] - 1);
00720         optimize->value[i * optimize->nvariables + j] = e;
00721     }
00722 }
00723 optimize->nsaveds = 0;
00724 if (nthreads <= 1)
00725     optimize_sequential ();
00726 else
00727 {
00728     for (i = 0; i < nthreads; ++i)
00729     {
00730         data[i].thread = i;
00731         thread[i]
00732             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00733     }
00734     for (i = 0; i < nthreads; ++i)
00735         g_thread_join (thread[i]);
00736 }
00737 #if HAVE_MPI
00738     // Communicating tasks results
00739     optimize_synchronise ();
00740 #endif
00741 #if DEBUG_OPTIMIZE
00742     fprintf (stderr, "optimize_sweep: end\n");
00743 #endif
00744 }
00745
00750 void
00751 optimize_MonteCarlo ()
00752 {
00753     unsigned int i, j;
00754     GThread *thread[nthreads];
00755     ParallelData data[nthreads];
00756 #if DEBUG_OPTIMIZE
00757     fprintf (stderr, "optimize_MonteCarlo: start\n");
00758 #endif
00759     for (i = 0; i < optimize->nsimulations; ++i)
00760         for (j = 0; j < optimize->nvariables; ++j)
00761             optimize->value[i * optimize->nvariables + j]
00762                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00763                     * (optimize->rangemax[j] - optimize->rangemin[j]);
00764     optimize->nsaveds = 0;
00765     if (nthreads <= 1)
00766         optimize_sequential ();
00767     else
00768     {
00769         for (i = 0; i < nthreads; ++i)
00770         {
00771             data[i].thread = i;
00772             thread[i]
00773                 = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00774         }
00775         for (i = 0; i < nthreads; ++i)
00776             g_thread_join (thread[i]);
00777     }
00778 #if HAVE_MPI
00779     // Communicating tasks results
00780     optimize_synchronise ();
00781 #endif
00782 #if DEBUG_OPTIMIZE
00783     fprintf (stderr, "optimize_MonteCarlo: end\n");
00784 #endif
00785 }
00786
00796 void
00797 optimize_best_direction (unsigned int simulation, double value)
00798 {
00799 #if DEBUG_OPTIMIZE
00800     fprintf (stderr, "optimize_best_direction: start\n");
00801     fprintf (stderr,
00802         "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00803         simulation, value, optimize->error_best[0]);
00804 #endif
00805     if (value < optimize->error_best[0])
00806     {
00807         optimize->error_best[0] = value;
00808         optimize->simulation_best[0] = simulation;
00809 #if DEBUG_OPTIMIZE
00810         fprintf (stderr,
00811             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00812             simulation, value);
00813 #endif
00814     }
00815 #if DEBUG_OPTIMIZE

```

```

00816     fprintf (stderr, "optimize_best_direction: end\n");
00817 #endif
00818 }
00819
00820 void
00821 optimize_direction_sequential (unsigned int simulation)
00822 {
00823     unsigned int i, j;
00824     double e;
00825 #if DEBUG_OPTIMIZE
00826     fprintf (stderr, "optimize_direction_sequential: start\n");
00827     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00828             "nend_direction=%u\n",
00829             optimize->nstart_direction, optimize->nend_direction);
00830 #endif
00831     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00832     {
00833         j = simulation + i;
00834         e = optimize_norm (j);
00835         optimize_best_direction (j, e);
00836         optimize_save_variables (j, e);
00837         if (e < optimize->threshold)
00838         {
00839             optimize->stop = 1;
00840             break;
00841         }
00842 #if DEBUG_OPTIMIZE
00843     fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00844 #endif
00845     }
00846 #if DEBUG_OPTIMIZE
00847     fprintf (stderr, "optimize_direction_sequential: end\n");
00848 #endif
00849 }
00850
00851 void *
00852 optimize_direction_thread (ParallelData * data)
00853 {
00854     unsigned int i, thread;
00855     double e;
00856 #if DEBUG_OPTIMIZE
00857     fprintf (stderr, "optimize_direction_thread: start\n");
00858 #endif
00859     thread = data->thread;
00860 #if DEBUG_OPTIMIZE
00861     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00862             thread,
00863             optimize->thread_direction[thread],
00864             optimize->thread_direction[thread + 1]);
00865 #endif
00866     for (i = optimize->thread_direction[thread];
00867          i < optimize->thread_direction[thread + 1]; ++i)
00868     {
00869         e = optimize_norm (i);
00870         g_mutex_lock (mutex);
00871         optimize_best_direction (i, e);
00872         optimize_save_variables (i, e);
00873         if (e < optimize->threshold)
00874         {
00875             optimize->stop = 1;
00876             g_mutex_unlock (mutex);
00877             if (optimize->stop)
00878                 break;
00879 #if DEBUG_OPTIMIZE
00880             fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00881 #endif
00882         }
00883 #if DEBUG_OPTIMIZE
00884     fprintf (stderr, "optimize_direction_thread: end\n");
00885 #endif
00886     g_thread_exit (NULL);
00887     return NULL;
00888 }
00889
00890 double
00891 optimize_estimate_direction_random (unsigned int variable,
00892                                     unsigned int estimate)
00893 {
00894     double x;
00895 #if DEBUG_OPTIMIZE
00896     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00897 #endif
00898     x = optimize->direction[variable]
00899         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00900 #if DEBUG_OPTIMIZE
00901     fprintf (stderr, "optimize_estimate_direction_random: direction=%u=%lg\n",
00902             variable, x);
00903     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00904 #endif

```

```

00925 #endif
00926     return x;
00927 }
00928
00929 double
00930 optimize_estimate_direction_coordinates (unsigned int variable,
00940                                         unsigned int estimate)
00941 {
00942     double x;
00943     #if DEBUG_OPTIMIZE
00944         fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00945     #endif
00946     x = optimize->direction[variable];
00947     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00948     {
00949         if (estimate & 1)
00950             x += optimize->step[variable];
00951         else
00952             x -= optimize->step[variable];
00953     }
00954     #if DEBUG_OPTIMIZE
00955         fprintf (stderr,
00956                 "optimize_estimate_direction_coordinates: direction=%lg\n",
00957                 variable, x);
00958         fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00959     #endif
00960     return x;
00961 }
00962
00963 void
00964 optimize_step_direction (unsigned int simulation)
00965 {
00966     GThread *thread[nthreads_direction];
00967     ParallelData data[nthreads_direction];
00968     unsigned int i, j, k, b;
00969     #if DEBUG_OPTIMIZE
00970         fprintf (stderr, "optimize_step_direction: start\n");
00971     #endif
00972     for (i = 0; i < optimize->nestimates; ++i)
00973     {
00974         k = (simulation + i) * optimize->nvariables;
00975         b = optimize->simulation_best[0] * optimize->nvariables;
00976         #if DEBUG_OPTIMIZE
00977             fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00978                     simulation + i, optimize->simulation_best[0]);
00979         #endif
00980         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00981         {
00982             #if DEBUG_OPTIMIZE
00983                 fprintf (stderr,
00984                         "optimize_step_direction: estimate=%u best=%u=%.14le\n",
00985                         i, j, optimize->value[b]);
00986             #endif
00987             optimize->value[k]
00988                 = optimize->value[b] + optimize_estimate_direction (j, i);
00989             optimize->value[k] = fmin (fmax (optimize->value[k],
00990                                             optimize->rangeminabs[j]),
00991                                     optimize->rangemaxabs[j]);
00992             #if DEBUG_OPTIMIZE
00993                 fprintf (stderr,
00994                         "optimize_step_direction: estimate=%u variable=%u=%.14le\n",
00995                         i, j, optimize->value[k]);
00996             #endif
00997         }
00998     }
00999     if (nthreads_direction == 1)
01000         optimize_direction_sequential (simulation);
01001     else
01002     {
01003         for (i = 0; i <= nthreads_direction; ++i)
01004         {
01005             optimize->thread_direction[i]
01006                 = simulation + optimize->nstart_direction
01007                   + i * (optimize->end_direction - optimize->
01008                        nstart_direction)
01009                   / nthreads_direction;
01010             #if DEBUG_OPTIMIZE
01011                 fprintf (stderr,
01012                         "optimize_step_direction: i=%u thread_direction=%u\n",
01013                         i, optimize->thread_direction[i]);
01014             #endif
01015         }
01016         for (i = 0; i < nthreads_direction; ++i)
01017         {
01018             data[i].thread = i;
01019             thread[i] = g_thread_new
01020                 (NULL, (GThreadFunc) optimize_direction_thread, &data[i]);
01021         }
01022     }
01023 }

```

```

01026     }
01027     for (i = 0; i < nthreads_direction; ++i)
01028         g_thread_join (thread[i]);
01029 }
01030 #if DEBUG_OPTIMIZE
01031 fprintf (stderr, "optimize_step_direction: end\n");
01032 #endif
01033 }
01034
01035 void
01040 optimize_direction ()
01041 {
01042     unsigned int i, j, k, b, s, adjust;
01043     #if DEBUG_OPTIMIZE
01044         fprintf (stderr, "optimize_direction: start\n");
01045     #endif
01046     for (i = 0; i < optimize->nvariables; ++i)
01047         optimize->direction[i] = 0.;
01048     b = optimize->simulation_best[0] * optimize->nvariables;
01049     s = optimize->nsimulations;
01050     adjust = 1;
01051     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01052     {
01053         #if DEBUG_OPTIMIZE
01054             fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01055                     i, optimize->simulation_best[0]);
01056         #endif
01057         optimize_step_direction (s);
01058         k = optimize->simulation_best[0] * optimize->nvariables;
01059         #if DEBUG_OPTIMIZE
01060             fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01061                     i, optimize->simulation_best[0]);
01062         #endif
01063         if (k == b)
01064         {
01065             if (adjust)
01066                 for (j = 0; j < optimize->nvariables; ++j)
01067                     optimize->step[j] *= 0.5;
01068             for (j = 0; j < optimize->nvariables; ++j)
01069                 optimize->direction[j] = 0.;
01070             adjust = 1;
01071         }
01072         else
01073         {
01074             for (j = 0; j < optimize->nvariables; ++j)
01075             {
01076                 #if DEBUG_OPTIMIZE
01077                     fprintf (stderr,
01078                             "optimize_direction: best=%u=%.14le old=%u=%.14le\n",
01079                             j, optimize->value[k + j], j, optimize->value[b + j]);
01080                 #endif
01081                 optimize->direction[j]
01082                     = (1. - optimize->relaxation) * optimize->direction[j]
01083                     + optimize->relaxation
01084                     * (optimize->value[k + j] - optimize->value[b + j]);
01085                 #if DEBUG_OPTIMIZE
01086                     fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01087                             j, optimize->direction[j]);
01088                 #endif
01089             }
01090             adjust = 0;
01091         }
01092     }
01093     #if DEBUG_OPTIMIZE
01094         fprintf (stderr, "optimize_direction: end\n");
01095     #endif
01096 }
01097
01105 double
01106 optimize_genetic_objective (Entity * entity)
01107 {
01108     unsigned int j;
01109     double objective;
01110     char buffer[64];
01111     #if DEBUG_OPTIMIZE
01112         fprintf (stderr, "optimize_genetic_objective: start\n");
01113     #endif
01114     for (j = 0; j < optimize->nvariables; ++j)
01115     {
01116         optimize->value[entity->id * optimize->nvariables + j]
01117             = genetic_get_variable (entity, optimize->genetic_variable + j);
01118     }
01119     objective = optimize_norm (entity->id);
01120     g_mutex_lock (mutex);
01121     for (j = 0; j < optimize->nvariables; ++j)
01122     {
01123         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);

```

```

01124     fprintf (optimize->file_variables, buffer,
01125               genetic_get_variable (entity, optimize->genetic_variable + j));
01126 }
01127 fprintf (optimize->file_variables, "%.14le\n", objective);
01128 g_mutex_unlock (mutex);
01129 #if DEBUG_OPTIMIZE
01130 fprintf (stderr, "optimize_genetic_objective: end\n");
01131 #endif
01132 return objective;
01133 }
01134
01139 void
01140 optimize_genetic ()
01141 {
01142     char *best_genome;
01143     double best_objective, *best_variable;
01144     #if DEBUG_OPTIMIZE
01145     fprintf (stderr, "optimize_genetic: start\n");
01146     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01147             nthreads);
01148     fprintf (stderr,
01149             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01150             optimize->nvariables, optimize->nsimulations, optimize->
01151             niterations);
01152     fprintf (stderr,
01153             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01154             optimize->mutation_ratio, optimize->reproduction_ratio,
01155             optimize->adaptation_ratio);
01156     #endif
01157     genetic_algorithm_default (optimize->nvariables,
01158                               optimize->genetic_variable,
01159                               optimize->nsimulations,
01160                               optimize->niterations,
01161                               optimize->mutation_ratio,
01162                               optimize->reproduction_ratio,
01163                               optimize->adaptation_ratio,
01164                               optimize->seed,
01165                               optimize->threshold,
01166                               &optimize_genetic_objective,
01167                               &best_genome, &best_variable, &best_objective);
01168     #if DEBUG_OPTIMIZE
01169     fprintf (stderr, "optimize_genetic: the best\n");
01170     #endif
01171     optimize->error_old = (double *) g_malloc (sizeof (double));
01172     optimize->value_old
01173     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01174     optimize->error_old[0] = best_objective;
01175     memcpy (optimize->value_old, best_variable,
01176            optimize->nvariables * sizeof (double));
01177     g_free (best_genome);
01178     g_free (best_variable);
01179     optimize_print ();
01180     #if DEBUG_OPTIMIZE
01181     fprintf (stderr, "optimize_genetic: end\n");
01182     #endif
01183 }
01184
01188 void
01189 optimize_save_old ()
01190 {
01191     unsigned int i, j;
01192     #if DEBUG_OPTIMIZE
01193     fprintf (stderr, "optimize_save_old: start\n");
01194     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01195     #endif
01196     memcpy (optimize->error_old, optimize->error_best,
01197            optimize->nbest * sizeof (double));
01198     for (i = 0; i < optimize->nbest; ++i)
01199     {
01200         j = optimize->simulation_best[i];
01201         #if DEBUG_OPTIMIZE
01202         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01203         #endif
01204         memcpy (optimize->value_old + i * optimize->nvariables,
01205                optimize->value + j * optimize->nvariables,
01206                optimize->nvariables * sizeof (double));
01207     }
01208     #if DEBUG_OPTIMIZE
01209     for (i = 0; i < optimize->nvariables; ++i)
01210         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01211                 i, optimize->value_old[i]);
01212     fprintf (stderr, "optimize_save_old: end\n");
01213     #endif
01214 }
01215
01221 void
01222 optimize_merge_old ()

```



```

01223 {
01224     unsigned int i, j, k;
01225     double v[optimize->nbest * optimize->nvariables], e[optimize->
nbest],
01226         *enew, *eold;
01227     #if DEBUG_OPTIMIZE
01228     fprintf (stderr, "optimize_merge_old: start\n");
01229     #endif
01230     anew = 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->
nvariables,
01240                     optimize->nvariables * sizeof (double));
01241             e[k] = *enew;
01242             ++k;
01243             ++enew;
01244             ++i;
01245         }
01246         else
01247         {
01248             memcpy (v + k * optimize->nvariables,
01249                     optimize->value_old + j * optimize->nvariables,
01250                     optimize->nvariables * sizeof (double));
01251             e[k] = *eold;
01252             ++k;
01253             ++eold;
01254             ++j;
01255         }
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
01270 void
01271 optimize_refine ()
01272 {
01273     unsigned int i, j;
01274     double d;
01275     #if HAVE_MPI
01276     MPI_Status mpi_stat;
01277     #endif
01278     #if DEBUG_OPTIMIZE
01279     fprintf (stderr, "optimize_refine: start\n");
01280     #endif
01281     #if HAVE_MPI
01282     if (!optimize->mpi_rank)
01283     {
01284         #endif
01285         for (j = 0; j < optimize->nvariables; ++j)
01286         {
01287             optimize->rangemin[j] = optimize->rangemax[j]
= 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]
= fmin (optimize->rangemin[j],
01294          optimize->value_old[i * optimize->nvariables + j]);
01295                 optimize->rangemax[j]
= fmax (optimize->rangemax[j],
01296          optimize->value_old[i * optimize->nvariables + j]);
01297             }
01298         }
01299         for (j = 0; j < optimize->nvariables; ++j)
01300         {
01301             d = optimize->tolerance
* (optimize->rangemax[j] - optimize->rangemin[j]);
01302             switch (optimize->algorithm)
01303             {
01304                 case ALGORITHM_MONTE_CARLO:
01305                     d *= 0.5;
01306                     break;
01307                 default:
01308                     if (optimize->nsweeps[j] > 1)
01309

```

```

01313         d /= optimize->nsweeps[j] - 1;
01314     else
01315         d = 0.;
01316     }
01317     optimize->rangemin[j] -= d;
01318     optimize->rangemin[j]
01319     = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01320     optimize->rangemax[j] += d;
01321     optimize->rangemax[j]
01322     = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01323     printf ("%s min=%lg max=%lg\n", optimize->label[j],
01324            optimize->rangemin[j], optimize->rangemax[j]);
01325     fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01326            optimize->label[j], optimize->rangemin[j],
01327            optimize->rangemax[j]);
01328     }
01329     #if HAVE_MPI
01330     for (i = 1; i < ntasks; ++i)
01331     {
01332         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01333                 1, MPI_COMM_WORLD);
01334         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01335                 1, MPI_COMM_WORLD);
01336     }
01337     }
01338     else
01339     {
01340         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01341                 MPI_COMM_WORLD, &mpi_stat);
01342         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01343                 MPI_COMM_WORLD, &mpi_stat);
01344     }
01345     #endif
01346     #if DEBUG_OPTIMIZE
01347     fprintf (stderr, "optimize_refine: end\n");
01348     #endif
01349     }
01350
01351     void
01352     optimize_step ()
01353     {
01354         #if DEBUG_OPTIMIZE
01355         fprintf (stderr, "optimize_step: start\n");
01356         #endif
01357         optimize_algorithm ();
01358         if (optimize->nsteps)
01359             optimize_direction ();
01360         #if DEBUG_OPTIMIZE
01361         fprintf (stderr, "optimize_step: end\n");
01362         #endif
01363     }
01364
01365     void
01366     optimize_iterate ()
01367     {
01368         unsigned int i;
01369         #if DEBUG_OPTIMIZE
01370         fprintf (stderr, "optimize_iterate: start\n");
01371         #endif
01372         optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01373         optimize->value_old =
01374             (double *) g_malloc (optimize->nbest * optimize->nvariables *
01375                                 sizeof (double));
01376         optimize_step ();
01377         optimize_save_old ();
01378         optimize_refine ();
01379         optimize_print ();
01380         for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01381         {
01382             optimize_step ();
01383             optimize_merge_old ();
01384             optimize_refine ();
01385             optimize_print ();
01386         }
01387         #if DEBUG_OPTIMIZE
01388         fprintf (stderr, "optimize_iterate: end\n");
01389         #endif
01390     }
01391
01392     void
01393     optimize_free ()
01394     {
01395         unsigned int i, j;
01396         #if DEBUG_OPTIMIZE
01397         fprintf (stderr, "optimize_free: start\n");
01398         #endif
01399         for (j = 0; j < optimize->ninputs; ++j)

```

```

01412     {
01413         for (i = 0; i < optimize->nexperiments; ++i)
01414             g_mapped_file_unref (optimize->file[j][i]);
01415         g_free (optimize->file[j]);
01416     }
01417     g_free (optimize->error_old);
01418     g_free (optimize->value_old);
01419     g_free (optimize->value);
01420     g_free (optimize->genetic_variable);
01421 #if DEBUG_OPTIMIZE
01422     fprintf (stderr, "optimize_free: end\n");
01423 #endif
01424 }
01425
01430 void
01431 optimize_open ()
01432 {
01433     GTimeZone *tz;
01434     GDateTime *t0, *t;
01435     unsigned int i, j;
01436
01437 #if DEBUG_OPTIMIZE
01438     char *buffer;
01439     fprintf (stderr, "optimize_open: start\n");
01440 #endif
01441
01442     // Getting initial time
01443 #if DEBUG_OPTIMIZE
01444     fprintf (stderr, "optimize_open: getting initial time\n");
01445 #endif
01446     tz = g_time_zone_new_utc ();
01447     t0 = g_date_time_new_now (tz);
01448
01449     // Obtaining and initing the pseudo-random numbers generator seed
01450 #if DEBUG_OPTIMIZE
01451     fprintf (stderr, "optimize_open: getting initial seed\n");
01452 #endif
01453     if (optimize->seed == DEFAULT_RANDOM_SEED)
01454         optimize->seed = input->seed;
01455     gsl_rng_set (optimize->rng, optimize->seed);
01456
01457     // Replacing the working directory
01458 #if DEBUG_OPTIMIZE
01459     fprintf (stderr, "optimize_open: replacing the working directory\n");
01460 #endif
01461     g_chdir (input->directory);
01462
01463     // Getting results file names
01464     optimize->result = input->result;
01465     optimize->variables = input->variables;
01466
01467     // Obtaining the simulator file
01468     optimize->simulator = input->simulator;
01469
01470     // Obtaining the evaluator file
01471     optimize->evaluator = input->evaluator;
01472
01473     // Reading the algorithm
01474     optimize->algorithm = input->algorithm;
01475     switch (optimize->algorithm)
01476     {
01477         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     #if DEBUG_OPTIMIZE
01514     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01515     #endif
01516     optimize->simulation_best
01517         = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01518     optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01519
01520     // Reading the experimental data
01521     #if DEBUG_OPTIMIZE
01522     buffer = g_get_current_dir ();
01523     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01524     g_free (buffer);
01525     #endif
01526     optimize->nexperiments = input->nexperiments;
01527     optimize->ninputs = input->experiment->ninputs;
01528     optimize->experiment
01529         = (char **) alloca (input->nexperiments * sizeof (char *));
01530     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01531     for (i = 0; i < input->experiment->ninputs; ++i)
01532         optimize->file[i] = (GMappedFile **)
01533             g_malloc (input->nexperiments * sizeof (GMappedFile *));
01534     for (i = 0; i < input->nexperiments; ++i)
01535     {
01536         #if DEBUG_OPTIMIZE
01537         fprintf (stderr, "optimize_open: i=%u\n", i);
01538         #endif
01539         optimize->experiment[i] = input->experiment[i].
01540             name;
01541         optimize->weight[i] = input->experiment[i].weight;
01542         #if DEBUG_OPTIMIZE
01543         fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01544             optimize->experiment[i], optimize->weight[i]);
01545         #endif
01546         for (j = 0; j < input->experiment->ninputs; ++j)
01547         {
01548             #if DEBUG_OPTIMIZE
01549             fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01550             #endif
01551             optimize->file[j][i]
01552                 = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01553         }
01554     }
01555     // Reading the variables data
01556     #if DEBUG_OPTIMIZE
01557     fprintf (stderr, "optimize_open: reading variables\n");
01558     #endif
01559     optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01560     j = input->nvariables * sizeof (double);
01561     optimize->rangemin = (double *) alloca (j);
01562     optimize->rangeminabs = (double *) alloca (j);
01563     optimize->rangemax = (double *) alloca (j);
01564     optimize->rangemaxabs = (double *) alloca (j);
01565     optimize->step = (double *) alloca (j);
01566     j = input->nvariables * sizeof (unsigned int);
01567     optimize->precision = (unsigned int *) alloca (j);
01568     optimize->nsweeps = (unsigned int *) alloca (j);
01569     optimize->nbits = (unsigned int *) alloca (j);
01570     for (i = 0; i < input->nvariables; ++i)
01571     {
01572         optimize->label[i] = input->variable[i].name;
01573         optimize->rangemin[i] = input->variable[i].rangemin;
01574         optimize->rangeminabs[i] = input->variable[i].
01575             rangeminabs;
01576         optimize->rangemax[i] = input->variable[i].rangemax;
01577         optimize->rangemaxabs[i] = input->variable[i].
01578             rangemaxabs;
01579         optimize->precision[i] = input->variable[i].
01580             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)

```

```

01583     {
01584         optimize->nsimulations = 1;
01585         for (i = 0; i < input->nvariables; ++i)
01586         {
01587             if (input->algorithm == ALGORITHM_SWEEP)
01588             {
01589                 optimize->nsimulations *= optimize->nsweeps[i];
01590 #if DEBUG_OPTIMIZE
01591                 fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01592                     optimize->nsweeps[i], optimize->nsimulations);
01593 #endif
01594             }
01595         }
01596     }
01597     if (optimize->nsteps)
01598         optimize->direction
01599         = (double *) alloca (optimize->nvariables * sizeof (double));
01600
01601     // Setting error norm
01602     switch (input->norm)
01603     {
01604         case ERROR_NORM_EUCLIDIAN:
01605             optimize_norm = optimize_norm_euclidian;
01606             break;
01607         case ERROR_NORM_MAXIMUM:
01608             optimize_norm = optimize_norm_maximum;
01609             break;
01610         case ERROR_NORM_P:
01611             optimize_norm = optimize_norm_p;
01612             optimize->p = input->p;
01613             break;
01614         default:
01615             optimize_norm = optimize_norm_taxicab;
01616     }
01617
01618     // Allocating values
01619 #if DEBUG_OPTIMIZE
01620     fprintf (stderr, "optimize_open: allocating variables\n");
01621     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01622         optimize->nvariables, optimize->algorithm);
01623 #endif
01624     optimize->genetic_variable = NULL;
01625     if (optimize->algorithm == ALGORITHM_GENETIC)
01626     {
01627         optimize->genetic_variable = (GeneticVariable *)
01628             g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01629         for (i = 0; i < optimize->nvariables; ++i)
01630         {
01631 #if DEBUG_OPTIMIZE
01632             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01633                 i, optimize->rangemin[i], optimize->rangemax[i],
01634                 optimize->nbits[i]);
01635 #endif
01636             optimize->genetic_variable[i].minimum = optimize->
01637                 rangemin[i];
01638             optimize->genetic_variable[i].maximum = optimize->
01639                 rangemax[i];
01640             optimize->genetic_variable[i].nbits = optimize->nbits[i];
01641         }
01642 #if DEBUG_OPTIMIZE
01643         fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01644             optimize->nvariables, optimize->nsimulations);
01645 #endif
01646         optimize->value = (double *)
01647             g_malloc ((optimize->nsimulations
01648                 + optimize->nestimates * optimize->nsteps)
01649                 * optimize->nvariables * sizeof (double));
01650
01651         // Calculating simulations to perform for each task
01652 #if HAVE_MPI
01653 #if DEBUG_OPTIMIZE
01654         fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01655             optimize->mpi_rank, ntasks);
01656 #endif
01657         optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01658             ntasks;
01659         optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01660             ntasks;
01661         if (optimize->nsteps)
01662         {
01663             optimize->nstart_direction
01664                 = optimize->mpi_rank * optimize->nestimates / ntasks;
01665             optimize->nend_direction
01666                 = (1 + optimize->mpi_rank) * optimize->nestimates /
01667                 ntasks;
01668         }
01669     }

```

```

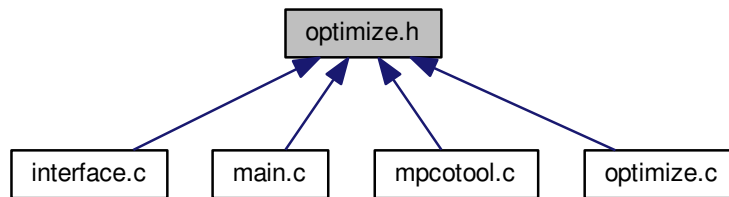
01665 #else
01666     optimize->nstart = 0;
01667     optimize->nend = optimize->nsimulations;
01668     if (optimize->nsteps)
01669     {
01670         optimize->nstart_direction = 0;
01671         optimize->nend_direction = optimize->nestimates;
01672     }
01673 #endif
01674 #if DEBUG_OPTIMIZE
01675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

```

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 *[stencil](#))
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**
- unsigned int **nthreads**
- unsigned int [nthreads_direction](#)
Number of threads for the direction search method.
- GMutex **mutex** [1]
- 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 2012-2017, 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 [470](#) of file [optimize.c](#).

```
00471 {
00472     unsigned int i, j;
00473     double e;
00474     #if DEBUG_OPTIMIZE
00475     fprintf (stderr, "optimize_best: start\n");
00476     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00477             optimize->nsaveds, optimize->nbest);
00478     #endif
00479     if (optimize->nsaveds < optimize->nbest
00480         || value < optimize->error_best[optimize->nsaveds - 1])
00481     {
00482         if (optimize->nsaveds < optimize->nbest)
00483             ++optimize->nsaveds;
00484         optimize->error_best[optimize->nsaveds - 1] = value;
00485         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00486         for (i = optimize->nsaveds; --i;)
00487         {
00488             if (optimize->error_best[i] < optimize->
00489                 error_best[i - 1])
00489             {
00490                 j = optimize->simulation_best[i];
00491                 e = optimize->error_best[i];
00492                 optimize->simulation_best[i] = optimize->
00493                     simulation_best[i - 1];
```

```

00493         optimize->error_best[i] = optimize->
error_best[i - 1];
00494         optimize->simulation_best[i - 1] = j;
00495         optimize->error_best[i - 1] = e;
00496     }
00497     else
00498         break;
00499 }
00500 }
00501 #if DEBUG_OPTIMIZE
00502 fprintf (stderr, "optimize_best: end\n");
00503 #endif
00504 }

```

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

```

00798 {
00799 #if DEBUG_OPTIMIZE
00800     fprintf (stderr, "optimize_best_direction: start\n");
00801     fprintf (stderr,
00802         "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00803         simulation, value, optimize->error_best[0]);
00804 #endif
00805     if (value < optimize->error_best[0])
00806     {
00807         optimize->error_best[0] = value;
00808         optimize->simulation_best[0] = simulation;
00809 #if DEBUG_OPTIMIZE
00810         fprintf (stderr,
00811             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00812             simulation, value);
00813 #endif
00814     }
00815 #if DEBUG_OPTIMIZE
00816     fprintf (stderr, "optimize_best_direction: end\n");
00817 #endif
00818 }

```

4.19.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

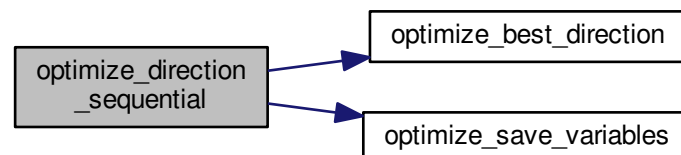
<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 827 of file [optimize.c](#).

```

00828 {
00829     unsigned int i, j;
00830     double e;
00831     #if DEBUG_OPTIMIZE
00832     fprintf (stderr, "optimize_direction_sequential: start\n");
00833     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00834             "nend_direction=%u\n",
00835             optimize->nstart_direction, optimize->
nend_direction);
00836     #endif
00837     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00838     {
00839         j = simulation + i;
00840         e = optimize_norm (j);
00841         optimize_best_direction (j, e);
00842         optimize_save_variables (j, e);
00843         if (e < optimize->threshold)
00844         {
00845             optimize->stop = 1;
00846             break;
00847         }
00848     #if DEBUG_OPTIMIZE
00849     fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00850     #endif
00851     }
00852     #if DEBUG_OPTIMIZE
00853     fprintf (stderr, "optimize_direction_sequential: end\n");
00854     #endif
00855 }
```

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

```

00866 {
00867     unsigned int i, thread;
00868     double e;
00869     #if DEBUG_OPTIMIZE
00870     fprintf (stderr, "optimize_direction_thread: start\n");
00871     #endif
00872     thread = data->thread;
00873     #if DEBUG_OPTIMIZE
00874     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00875             thread,
00876             optimize->thread_direction[thread],
00877             optimize->thread_direction[thread + 1]);
00878     #endif
00879     for (i = optimize->thread_direction[thread];
00880          i < optimize->thread_direction[thread + 1]; ++i)
00881     {
00882         e = optimize_norm (i);
00883         g_mutex_lock (mutex);
00884         optimize_best_direction (i, e);
00885         optimize_save_variables (i, e);
00886         if (e < optimize->threshold)
00887             optimize->stop = 1;
00888         g_mutex_unlock (mutex);
00889         if (optimize->stop)
00890             break;
00891     #if DEBUG_OPTIMIZE
00892     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00893     #endif
00894     }
00895     #if DEBUG_OPTIMIZE
00896     fprintf (stderr, "optimize_direction_thread: end\n");
00897     #endif
00898     g_thread_exit (NULL);
00899     return NULL;
00900 }

```

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

```

00941 {
00942     double x;
00943     #if DEBUG_OPTIMIZE
00944     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00945     #endif
00946     x = optimize->direction[variable];
00947     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00948     {
00949         if (estimate & 1)
00950             x += optimize->step[variable];
00951         else
00952             x -= optimize->step[variable];
00953     }
00954     #if DEBUG_OPTIMIZE
00955     fprintf (stderr,
00956             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00957             variable, x);
00958     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00959     #endif
00960     return x;
00961 }

```

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

```

00914 {
00915     double x;
00916     #if DEBUG_OPTIMIZE
00917     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00918     #endif
00919     x = optimize->direction[variable]
00920         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00921         step[variable];
00922     #if DEBUG_OPTIMIZE
00923     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00924             variable, x);
00925     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00926     #endif
00927     return x;
00928 }

```

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

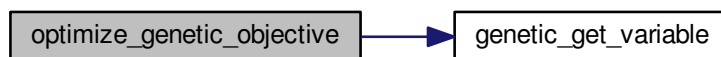
objective function value.

Definition at line 1106 of file [optimize.c](#).

```

01107 {
01108     unsigned int j;
01109     double objective;
01110     char buffer[64];
01111     #if DEBUG_OPTIMIZE
01112     fprintf (stderr, "optimize_genetic_objective: start\n");
01113     #endif
01114     for (j = 0; j < optimize->nvariables; ++j)
01115     {
01116         optimize->value[entity->id * optimize->nvariables + j]
01117             = genetic_get_variable (entity, optimize->genetic_variable + j);
01118     }
01119     objective = optimize_norm (entity->id);
01120     g_mutex_lock (mutex);
01121     for (j = 0; j < optimize->nvariables; ++j)
01122     {
01123         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01124         fprintf (optimize->file_variables, buffer,
01125                 genetic_get_variable (entity, optimize->genetic_variable + j));
01126     }
01127     fprintf (optimize->file_variables, "%.14le\n", objective);
01128     g_mutex_unlock (mutex);
01129     #if DEBUG_OPTIMIZE
01130     fprintf (stderr, "optimize_genetic_objective: end\n");
01131     #endif
01132     return objective;
01133 }
```

Here is the call graph for this function:



4.19.2.8 optimize_input()

```

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

Function to write the simulation input file.

Parameters

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

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

```

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

```

```

00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177     return;
00178 }

```

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

```

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

```



```

00638     optimize->nsaveds = k;
00639     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00640     memcpy (optimize->error_best, e, k * sizeof (double));
00641     #if DEBUG_OPTIMIZE
00642     fprintf (stderr, "optimize_merge: end\n");
00643     #endif
00644 }

```

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.

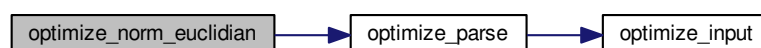
Definition at line [302](#) of file [optimize.c](#).

```

00303 {
00304     double e, ei;
00305     unsigned int i;
00306     #if DEBUG_OPTIMIZE
00307     fprintf (stderr, "optimize_norm_euclidian: start\n");
00308     #endif
00309     e = 0.;
00310     for (i = 0; i < optimize->nexperiments; ++i)
00311     {
00312         ei = optimize_parse (simulation, i);
00313         e += ei * ei;
00314     }
00315     e = sqrt (e);
00316     #if DEBUG_OPTIMIZE
00317     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00318     fprintf (stderr, "optimize_norm_euclidian: end\n");
00319     #endif
00320     return e;
00321 }

```

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.
-------------------	--------------------

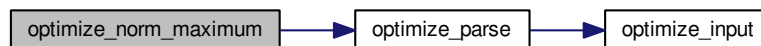
Returns

Maximum error norm.

Definition at line 331 of file [optimize.c](#).

```
00332 {
00333     double e, ei;
00334     unsigned int i;
00335     #if DEBUG_OPTIMIZE
00336     fprintf (stderr, "optimize_norm_maximum: start\n");
00337     #endif
00338     e = 0.;
00339     for (i = 0; i < optimize->nexperiments; ++i)
00340     {
00341         ei = fabs (optimize_parse (simulation, i));
00342         e = fmax (e, ei);
00343     }
00344     #if DEBUG_OPTIMIZE
00345     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00346     fprintf (stderr, "optimize_norm_maximum: end\n");
00347     #endif
00348     return e;
00349 }
```

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.
-------------------	--------------------

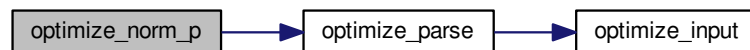
Returns

P error norm.

Definition at line 359 of file [optimize.c](#).

```
00360 {
00361     double e, ei;
00362     unsigned int i;
00363     #if DEBUG_OPTIMIZE
00364     fprintf (stderr, "optimize_norm_p: start\n");
00365     #endif
00366     e = 0.;
00367     for (i = 0; i < optimize->nexperiments; ++i)
00368     {
00369         ei = fabs (optimize_parse (simulation, i));
00370         e += pow (ei, optimize->p);
00371     }
00372     e = pow (e, 1. / optimize->p);
00373     #if DEBUG_OPTIMIZE
00374     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00375     fprintf (stderr, "optimize_norm_p: end\n");
00376     #endif
00377     return e;
00378 }
```

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

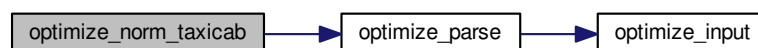
Taxicab error norm.

Definition at line 388 of file [optimize.c](#).

```

00389 {
00390     double e;
00391     unsigned int i;
00392     #if DEBUG_OPTIMIZE
00393     fprintf (stderr, "optimize_norm_taxicab: start\n");
00394     #endif
00395     e = 0.;
00396     for (i = 0; i < optimize->nexperiments; ++i)
00397         e += fabs (optimize_parse (simulation, i));
00398     #if DEBUG_OPTIMIZE
00399     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00400     fprintf (stderr, "optimize_norm_taxicab: end\n");
00401     #endif
00402     return e;
00403 }
```

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, experiment);
00209     #if DEBUG_OPTIMIZE
00210         fprintf(stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00211     #endif
00212         optimize_input (simulation, &input[i][0], optimize->
00213             file[i][experiment]);
00214     }
00215     for (; i < MAX_NINPUTS; ++i)
00216         strcpy (&input[i][0], "");
00217     #if DEBUG_OPTIMIZE
00218     fprintf(stderr, "optimize_parse: parsing end\n");
00219     #endif
00220
00221     // Performing the simulation
00222     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00223     buffer2 = g_path_get_dirname (optimize->simulator);
00224     buffer3 = g_path_get_basename (optimize->simulator);
00225     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00226     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00227             buffer4, input[0], input[1], input[2], input[3], input[4],
00228             input[5], input[6], input[7], output);
00229     g_free (buffer4);
00230     g_free (buffer3);
00231     g_free (buffer2);
00232     #if DEBUG_OPTIMIZE
00233     fprintf(stderr, "optimize_parse: %s\n", buffer);
00234     #endif
00235     system (buffer);
00236
00237     // Checking the objective value function
00238     if (optimize->evaluator)
00239     {
00240         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00241         buffer2 = g_path_get_dirname (optimize->evaluator);
00242         buffer3 = g_path_get_basename (optimize->evaluator);
00243         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00244         snprintf (buffer, 512, "\"%s\" %s %s %s",
00245             buffer4, output, optimize->experiment[experiment], result);
00246         g_free (buffer4);
00247         g_free (buffer3);
00248         g_free (buffer2);
00249     #if DEBUG_OPTIMIZE
00250         fprintf(stderr, "optimize_parse: %s\n", buffer);
00251         fprintf(stderr, "optimize_parse: result=%s\n", result);
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     #if DEBUG_OPTIMIZE
00261         fprintf(stderr, "optimize_parse: output=%s\n", output);
00262     #endif
00263         strcpy (result, "");
00264         file_result = g_fopen (output, "r");
00265         e = atof (fgets (buffer, 512, file_result));
00266         fclose (file_result);
00267     }
00268
00269     // Removing files
00270     #if !DEBUG_OPTIMIZE
00271     for (i = 0; i < optimize->ninputs; ++i)
00272     {
00273         if (optimize->file[i][0])
00274         {
00275             snprintf (buffer, 512, RM " %s", &input[i][0]);
00276             system (buffer);
00277         }
00278     }
00279     snprintf (buffer, 512, RM " %s %s", output, result);
00280     system (buffer);
00281     #endif

```

```

00281
00282 // Processing pending events
00283 if (show_pending)
00284     show_pending ();
00285
00286 #if DEBUG_OPTIMIZE
00287     fprintf (stderr, "optimize_parse: end\n");
00288 #endif
00289
00290 // Returning the objective function
00291 return e * optimize->weight[experiment];
00292 }

```

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

```

00442 {
00443     unsigned int i;
00444     char buffer[64];
00445     #if DEBUG_OPTIMIZE
00446     fprintf (stderr, "optimize_save_variables: start\n");
00447     #endif
00448     for (i = 0; i < optimize->nvariables; ++i)
00449     {
00450         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00451         fprintf (optimize->file_variables, buffer,
00452             optimize->value[simulation * optimize->
00453                 nvariables + i]);
00454     }
00455     fprintf (optimize->file_variables, "%.14le\n", error);
00456     fflush (optimize->file_variables);
00457     #if DEBUG_OPTIMIZE
00458     fprintf (stderr, "optimize_save_variables: end\n");
00459     #endif
00460 }

```

4.19.2.16 optimize_step_direction()

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

Function to do a step of the direction search method.

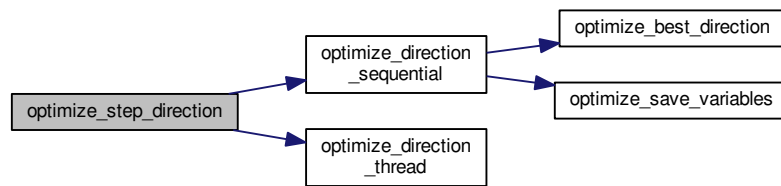
Parameters

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

Definition at line 970 of file [optimize.c](#).

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

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

```

00548 {
00549     unsigned int i, thread;
00550     double e;
00551     #if DEBUG_OPTIMIZE
00552     fprintf (stderr, "optimize_thread: start\n");
00553     #endif
00554     thread = data->thread;
00555     #if DEBUG_OPTIMIZE
00556     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00557             optimize->thread[thread], optimize->thread[thread + 1]);
00558     #endif
00559     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00560     {
00561         e = optimize_norm (i);
00562         g_mutex_lock (mutex);
00563         optimize_best (i, e);
00564         optimize_save_variables (i, e);
00565         if (e < optimize->threshold)
00566             optimize->stop = 1;
00567         g_mutex_unlock (mutex);
00568         if (optimize->stop)
00569             break;
00570     #if DEBUG_OPTIMIZE
00571     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00572     #endif
00573     }
00574     #if DEBUG_OPTIMIZE
00575     fprintf (stderr, "optimize_thread: end\n");
00576     #endif
00577     g_thread_exit (NULL);
00578     return NULL;
00579 }
```


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 2012-2017, 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 * stencil);
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.

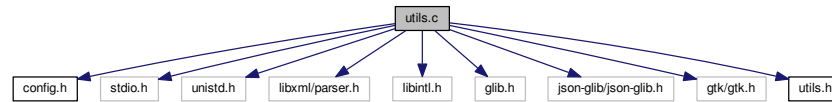
```

#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 2012-2017, 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 [531](#) of file [utils.c](#).

```
00532 {
00533 #ifdef G_OS_WIN32
00534     SYSTEM_INFO sysinfo;
00535     GetSystemInfo (&sysinfo);
00536     return sysinfo.dwNumberOfProcessors;
00537 #else
00538     return (int) sysconf (_SC_NPROCESSORS_ONLN);
00539 #endif
00540 }
```

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 566 of file [utils.c](#).

```
00567 {
00568     unsigned int i;
00569     for (i = 0; i < n; ++i)
00570         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00571             break;
00572     return i;
00573 }
```

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.

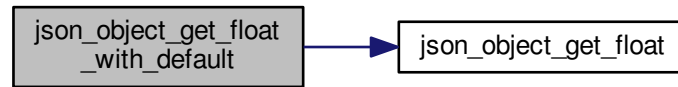
Returns

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 331 of file [utils.c](#).

```
00332 {  
00333     const char *buffer;  
00334     int i = 0;  
00335     buffer = json_object_get_string_member (object, prop);  
00336     if (!buffer)  
00337         *error_code = 1;  
00338     else  
00339     {  
00340         if (sscanf (buffer, "%d", &i) != 1)  
00341             *error_code = 2;  
00342         else  
00343             *error_code = 0;  
00344     }  
00345     return i;  
00346 }
```

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 361 of file [utils.c](#).

```
00362 {
00363     const char *buffer;
00364     unsigned int i = 0;
00365     buffer = json_object_get_string_member (object, prop);
00366     if (!buffer)
00367         *error_code = 1;
00368     else
00369     {
00370         if (sscanf (buffer, "%u", &i) != 1)
00371             *error_code = 2;
00372         else
00373             *error_code = 0;
00374     }
00375     return i;
00376 }
```

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.

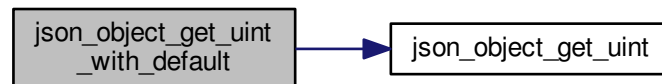
Returns

Unsigned integer number value.

Definition at line 394 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 518 of file [utils.c](#).

```
00519 {  
00520     char buffer[64];  
00521     snprintf (buffer, 64, "%.14lg", value);  
00522     json_object_set_string_member (object, prop, buffer);  
00523 }
```

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](#).

```
00500 {
00501     char buffer[64];
00502     snprintf (buffer, 64, "%u", value);
00503     json_object_set_string_member (object, prop, buffer);
00504 }
```

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 104 of file [utils.c](#).

```
00105 {
00106     show_message ( _("ERROR!"), msg, ERROR_TYPE);
00107 }
```

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

```

4.21.2.13 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 214 of file [utils.c](#).

```

00215 {
00216     double x = 0.;
00217     xmlChar *buffer;
00218     buffer = xmlGetProp (node, prop);
00219     if (!buffer)
00220         *error_code = 1;
00221     else
00222     {
00223         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00224             *error_code = 2;
00225         else
00226             *error_code = 0;
00227         xmlFree (buffer);
00228     }
00229     return x;
00230 }

```

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.

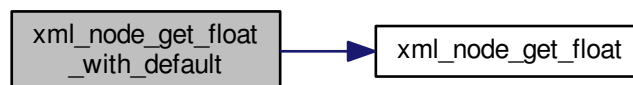
Returns

Floating point number value.

Definition at line 248 of file [utils.c](#).

```
00250 {
00251     double x;
00252     if (xmlHasProp (node, prop))
00253         x = xml_node_get_float (node, prop, error_code);
00254     else
00255     {
00256         x = default_value;
00257         *error_code = 0;
00258     }
00259     return x;
00260 }
```

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 122 of file [utils.c](#).

```
00123 {
00124     int i = 0;
00125     xmlChar *buffer;
00126     buffer = xmlGetProp (node, prop);
00127     if (!buffer)
00128         *error_code = 1;
00129     else
00130     {
00131         if (sscanf ((char *) buffer, "%d", &i) != 1)
00132             *error_code = 2;
00133         else
00134             *error_code = 0;
00135         xmlFree (buffer);
00136     }
00137     return i;
00138 }
```

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 153 of file [utils.c](#).

```
00154 {
00155     unsigned int i = 0;
00156     xmlChar *buffer;
00157     buffer = xmlGetProp (node, prop);
```

```

00158  if (!buffer)
00159      *error_code = 1;
00160  else
00161      {
00162          if (sscanf ((char *) buffer, "%u", &i) != 1)
00163              *error_code = 2;
00164          else
00165              *error_code = 0;
00166          xmlFree (buffer);
00167      }
00168  return i;
00169  }

```

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.

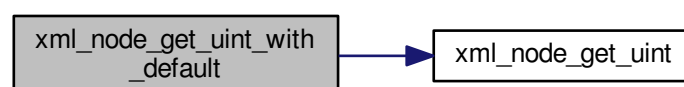
Definition at line 187 of file [utils.c](#).

```

00189 {
00190     unsigned int i;
00191     if (xmlHasProp (node, prop))
00192         i = xml_node_get_uint (node, prop, error_code);
00193     else
00194     {
00195         i = default_value;
00196         *error_code = 0;
00197     }
00198     return i;
00199 }

```

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 311 of file [utils.c](#).

```
00312 {
00313     xmlChar buffer[64];
00314     snprintf ((char *) buffer, 64, "%.14lg", value);
00315     xmlSetProp (node, prop, buffer);
00316 }
```

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 273 of file [utils.c](#).

```
00274 {
00275     xmlChar buffer[64];
00276     snprintf ((char *) buffer, 64, "%d", value);
00277     xmlSetProp (node, prop, buffer);
00278 }
```


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 292 of file [utils.c](#).

```
00293 {
00294     xmlChar buffer[64];
00295     snprintf ((char *) buffer, 64, "%u", value);
00296     xmlSetProp (node, prop, buffer);
00297 }
```

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 Burguete and Borja Latorre.
00007
00008 Copyright 2012-2017, 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
00062 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 *)
00080             gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00081                                     (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00082
00083         // Setting the dialog title
00084         gtk_window_set_title (GTK_WINDOW (dlg), title);
00085
00086         // Showing the dialog and waiting response
00087         gtk_dialog_run (GTK_DIALOG (dlg));
00088
00089         // Closing and freeing memory
00090         gtk_widget_destroy (GTK_WIDGET (dlg));
00091
00092     #else
00093         printf ("%s: %s\n", title, msg);
00094     #endif
00095 }
00096
00103 void
00104 show_error (char *msg)
00105 {
00106     show_message (_("ERROR!"), msg, ERROR_TYPE);
00107 }
00108
00121 int
00122 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00123 {
00124     int i = 0;
00125     xmlChar *buffer;
00126     buffer = xmlGetProp (node, prop);
00127     if (!buffer)
00128         *error_code = 1;
00129     else
00130     {
00131         if (sscanf ((char *) buffer, "%d", &i) != 1)
00132             *error_code = 2;
00133         else
00134             *error_code = 0;
00135         xmlFree (buffer);
00136     }
00137     return i;
00138 }
00139
00152 unsigned int
00153 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00154 {
00155     unsigned int i = 0;
00156     xmlChar *buffer;
00157     buffer = xmlGetProp (node, prop);
00158     if (!buffer)
00159         *error_code = 1;
00160     else
00161     {
00162         if (sscanf ((char *) buffer, "%u", &i) != 1)
00163             *error_code = 2;
00164         else
00165             *error_code = 0;
00166         xmlFree (buffer);
00167     }
00168     return i;
00169 }
00170
00186 unsigned int
00187 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00188                                 unsigned int default_value, int *error_code)
00189 {
00190     unsigned int i;
00191     if (xmlHasProp (node, prop))
00192         i = xml_node_get_uint (node, prop, error_code);
00193     else

```

```

00194     {
00195         i = default_value;
00196         *error_code = 0;
00197     }
00198     return i;
00199 }
00200
00213 double
00214 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00215 {
00216     double x = 0.;
00217     xmlChar *buffer;
00218     buffer = xmlGetProp (node, prop);
00219     if (!buffer)
00220         *error_code = 1;
00221     else
00222     {
00223         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00224             *error_code = 2;
00225         else
00226             *error_code = 0;
00227         xmlFree (buffer);
00228     }
00229     return x;
00230 }
00231
00247 double
00248 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00249                                 double default_value, int *error_code)
00250 {
00251     double x;
00252     if (xmlHasProp (node, prop))
00253         x = xml_node_get_float (node, prop, error_code);
00254     else
00255     {
00256         x = default_value;
00257         *error_code = 0;
00258     }
00259     return x;
00260 }
00261
00272 void
00273 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00274 {
00275     xmlChar buffer[64];
00276     snprintf ((char *) buffer, 64, "%d", value);
00277     xmlSetProp (node, prop, buffer);
00278 }
00279
00291 void
00292 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00293 {
00294     xmlChar buffer[64];
00295     snprintf ((char *) buffer, 64, "%u", value);
00296     xmlSetProp (node, prop, buffer);
00297 }
00298
00310 void
00311 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00312 {
00313     xmlChar buffer[64];
00314     snprintf ((char *) buffer, 64, "%.14lg", value);
00315     xmlSetProp (node, prop, buffer);
00316 }
00317
00330 int
00331 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00332 {
00333     const char *buffer;
00334     int i = 0;
00335     buffer = json_object_get_string_member (object, prop);
00336     if (!buffer)
00337         *error_code = 1;
00338     else
00339     {
00340         if (sscanf (buffer, "%d", &i) != 1)
00341             *error_code = 2;
00342         else
00343             *error_code = 0;
00344     }
00345     return i;
00346 }
00347
00360 unsigned int
00361 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00362 {
00363     const char *buffer;

```

```

00364 unsigned int i = 0;
00365 buffer = json_object_get_string_member (object, prop);
00366 if (!buffer)
00367     *error_code = 1;
00368 else
00369 {
00370     if (sscanf (buffer, "%u", &i) != 1)
00371         *error_code = 2;
00372     else
00373         *error_code = 0;
00374 }
00375 return i;
00376 }
00377
00393 unsigned int
00394 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00395                                   unsigned int default_value, 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                                   ,
00456                                   double default_value, int *error_code)
00457 {
00458     double x;
00459     if (json_object_get_member (object, prop))
00460         x = json_object_get_float (object, prop, error_code);
00461     else
00462     {
00463         x = default_value;
00464         *error_code = 0;
00465     }
00466     return x;
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, unsigned int value)
00500 {
00501     char buffer[64];
00502     snprintf (buffer, 64, "%u", value);
00503     json_object_set_string_member (object, prop, buffer);
00504 }
00505
00517 void
00518 json_object_set_float (JsonObject * object, const char *prop, double value)
00519 {
00520     char buffer[64];
00521     snprintf (buffer, 64, "%.14lg", value);
00522     json_object_set_string_member (object, prop, buffer);
00523 }
00524

```

```

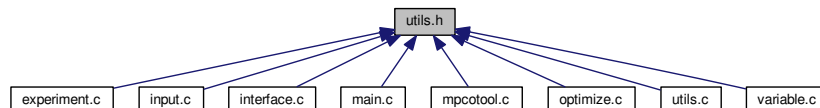
00530 int
00531 cores_number ()
00532 {
00533     #ifdef G_OS_WIN32
00534         SYSTEM_INFO sysinfo;
00535         GetSystemInfo (&sysinfo);
00536         return sysinfo.dwNumberOfProcessors;
00537     #else
00538         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00539     #endif
00540 }
00541
00542 #if HAVE_GTK
00543
00548 void
00549 process_pending ()
00550 {
00551     while (gtk_events_pending ())
00552         gtk_main_iteration ();
00553 }
00554
00565 unsigned int
00566 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00567 {
00568     unsigned int i;
00569     for (i = 0; i < n; ++i)
00570         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00571             break;
00572     return i;
00573 }
00574
00575 #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 2012-2017, 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 [531](#) of file [utils.c](#).

```
00532 {  
00533 #ifdef G_OS_WIN32  
00534     SYSTEM_INFO sysinfo;  
00535     GetSystemInfo (&sysinfo);  
00536     return sysinfo.dwNumberOfProcessors;  
00537 #else  
00538     return (int) sysconf (_SC_NPROCESSORS_ONLN);  
00539 #endif  
00540 }
```

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 566 of file [utils.c](#).

```

00567 {
00568     unsigned int i;
00569     for (i = 0; i < n; ++i)
00570         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00571             break;
00572     return i;
00573 }
```

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.

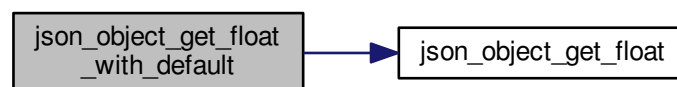
Returns

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 331 of file [utils.c](#).

```
00332 {
00333     const char *buffer;
00334     int i = 0;
00335     buffer = json_object_get_string_member (object, prop);
00336     if (!buffer)
00337         *error_code = 1;
00338     else
00339     {
00340         if (sscanf (buffer, "%d", &i) != 1)
00341             *error_code = 2;
00342         else
00343             *error_code = 0;
00344     }
00345     return i;
00346 }
```

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 361 of file [utils.c](#).

```
00362 {
00363     const char *buffer;
00364     unsigned int i = 0;
00365     buffer = json_object_get_string_member (object, prop);
```

```

00366     if (!buffer)
00367         *error_code = 1;
00368     else
00369     {
00370         if (sscanf (buffer, "%u", &i) != 1)
00371             *error_code = 2;
00372         else
00373             *error_code = 0;
00374     }
00375     return i;
00376 }

```

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.

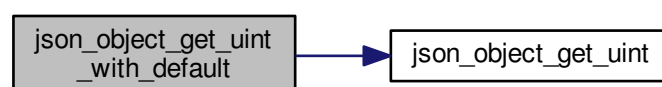
Definition at line 394 of file [utils.c](#).

```

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 518 of file [utils.c](#).

```
00519 {
00520     char buffer[64];
00521     snprintf (buffer, 64, "%.14lg", value);
00522     json_object_set_string_member (object, prop, buffer);
00523 }
```

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](#).

```
00500 {
00501     char buffer[64];
00502     snprintf (buffer, 64, "%u", value);
00503     json_object_set_string_member (object, prop, buffer);
00504 }
```

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 [104](#) of file [utils.c](#).

```
00105 {
00106     show_message (_("ERROR!"), msg, ERROR_TYPE);
00107 }
```

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

4.23.2.13 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 214 of file [utils.c](#).

```
00215 {
00216     double x = 0.;
00217     xmlChar *buffer;
00218     buffer = xmlGetProp (node, prop);
00219     if (!buffer)
00220         *error_code = 1;
00221     else
00222     {
00223         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00224             *error_code = 2;
00225         else
00226             *error_code = 0;
00227         xmlFree (buffer);
00228     }
00229     return x;
00230 }
```

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.

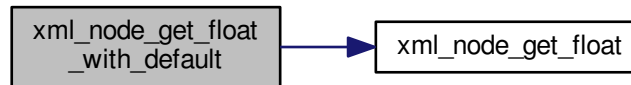
Returns

Floating point number value.

Definition at line 248 of file [utils.c](#).

```
00250 {
00251     double x;
00252     if (xmlHasProp (node, prop))
00253         x = xml_node_get_float (node, prop, error_code);
00254     else
00255     {
00256         x = default_value;
00257         *error_code = 0;
00258     }
00259     return x;
00260 }
```

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 122 of file [utils.c](#).

```

00123 {
00124     int i = 0;
00125     xmlChar *buffer;
00126     buffer = xmlGetProp (node, prop);
00127     if (!buffer)
00128         *error_code = 1;
00129     else
00130     {
00131         if (sscanf ((char *) buffer, "%d", &i) != 1)
00132             *error_code = 2;
00133         else
00134             *error_code = 0;
00135         xmlFree (buffer);
00136     }
00137     return i;
00138 }
  
```


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 153 of file [utils.c](#).

```
00154 {
00155     unsigned int i = 0;
00156     xmlChar *buffer;
00157     buffer = xmlGetProp (node, prop);
00158     if (!buffer)
00159         *error_code = 1;
00160     else
00161     {
00162         if (sscanf ((char *) buffer, "%u", &i) != 1)
00163             *error_code = 2;
00164         else
00165             *error_code = 0;
00166         xmlFree (buffer);
00167     }
00168     return i;
00169 }
```

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.

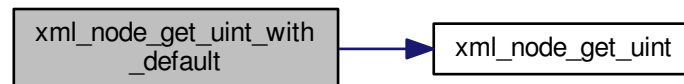
Definition at line 187 of file [utils.c](#).

```

00189 {
00190     unsigned int i;
00191     if (xmlHasProp (node, prop))
00192         i = xml_node_get_uint (node, prop, error_code);
00193     else
00194     {
00195         i = default_value;
00196         *error_code = 0;
00197     }
00198     return i;
00199 }

```

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 311 of file [utils.c](#).

```

00312 {
00313     xmlChar buffer[64];
00314     snprintf ((char *) buffer, 64, "%.14lg", value);
00315     xmlSetProp (node, prop, buffer);
00316 }

```

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 273 of file [utils.c](#).

```
00274 {
00275     xmlChar buffer[64];
00276     snprintf ((char *) buffer, 64, "%d", value);
00277     xmlSetProp (node, prop, buffer);
00278 }
```

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 292 of file [utils.c](#).

```
00293 {
00294     xmlChar buffer[64];
00295     snprintf ((char *) buffer, 64, "%u", value);
00296     xmlSetProp (node, prop, buffer);
00297 }
```

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 2012-2017, 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, int *error_code);
00061 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00062 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00063                        unsigned int value);
00064 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00065 int json_object_get_int (JsonObject * object, const char *prop,
00066                        int *error_code);
00067 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00068                                   int *error_code);
00069 unsigned int json_object_get_uint_with_default (JsonObject * object,
00070                                                const char *prop,
00071                                                unsigned int default_value,
00072                                                int *error_code);
00073 double json_object_get_float (JsonObject * object, const char *prop,
00074                              int *error_code);
00075 double json_object_get_float_with_default (JsonObject * object,
00076                                           const char *prop,
00077                                           double default_value,
00078                                           int *error_code);
00079 void json_object_set_int (JsonObject * object, const char *prop, int value);
00080 void json_object_set_uint (JsonObject * object, const char *prop,
00081                          unsigned int value);
00082 void json_object_set_float (JsonObject * object, const char *prop,
00083                          double value);
00084 int cores_number ();
00085 #if HAVE_GTK
00086 void process_pending ();
00087 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);

```

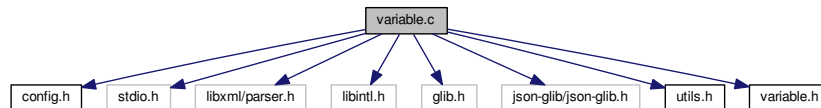
```
00100 #endif
00101
00102 #endif
```

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 2012-2017, 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, message);
00117     error_message = g_strdup (buffer);
00118 }
```

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 301 of file [variable.c](#).

```

00303 {
00304     JsonObject *object;
00305     const char *label;
00306     int error_code;
00307     #if DEBUG_VARIABLE
00308     fprintf (stderr, "variable_open_json: start\n");
00309     #endif
00310     object = json_node_get_object (node);
00311     label = json_object_get_string_member (object, LABEL_NAME);
00312     if (!label)
00313     {
00314         variable_error (variable, _("no name"));
00315         goto exit_on_error;
00316     }
00317     variable->name = g_strdup (label);
00318     if (json_object_get_member (object, LABEL_MINIMUM))
00319     {
00320         variable->rangemin
00321         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00322         if (error_code)
00323         {
00324             variable_error (variable, _("bad minimum"));
00325             goto exit_on_error;
00326         }
00327         variable->rangeminabs
00328         = json_object_get_float_with_default (object,
00329 LABEL_ABSOLUTE_MINIMUM,
00330                                             -G_MAXDOUBLE, &error_code);
00331         if (error_code)
00332         {
00333             variable_error (variable, _("bad absolute minimum"));
00334             goto exit_on_error;
00335         }
00336         if (variable->rangemin < variable->rangeminabs)
00337         {
00338             variable_error (variable, _("minimum range not allowed"));
00339             goto exit_on_error;
00340         }
00341     }
00342     else
00343     {
00344         variable_error (variable, _("no minimum range"));
00345         goto exit_on_error;
00346     }
00347     if (json_object_get_member (object, LABEL_MAXIMUM))
00348     {
00349         variable->rangemax
00350         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00351         if (error_code)
00352         {
00353             variable_error (variable, _("bad maximum"));
00354             goto exit_on_error;
00355         }
00356         variable->rangemaxabs
00357         = json_object_get_float_with_default (object,
00358 LABEL_ABSOLUTE_MAXIMUM,
00359                                             G_MAXDOUBLE, &error_code);
00360         if (error_code)
00361         {
00362             variable_error (variable, _("bad absolute maximum"));
00363             goto exit_on_error;
00364         }
00365         if (variable->rangemax > variable->rangemaxabs)
00366         {

```

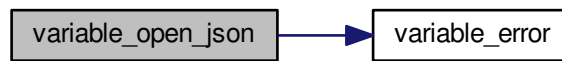


```

00365         variable_error (variable, _("maximum range not allowed"));
00366         goto exit_on_error;
00367     }
00368     if (variable->rangemax < variable->rangemin)
00369     {
00370         variable_error (variable, _("bad range"));
00371         goto exit_on_error;
00372     }
00373 }
00374 else
00375 {
00376     variable_error (variable, _("no maximum range"));
00377     goto exit_on_error;
00378 }
00379 variable->precision
00380 = json_object_get_uint_with_default (object,
LABEL_PRECISION,
00381                                     DEFAULT_PRECISION, &error_code);
00382 if (error_code || variable->precision >= NPRECISIONS)
00383 {
00384     variable_error (variable, _("bad precision"));
00385     goto exit_on_error;
00386 }
00387 if (algorithm == ALGORITHM_SWEEP)
00388 {
00389     if (json_object_get_member (object, LABEL_NSWEEPS))
00390     {
00391         variable->nsweeps
00392         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00393         if (error_code || !variable->nsweeps)
00394         {
00395             variable_error (variable, _("bad sweeps"));
00396             goto exit_on_error;
00397         }
00398     }
00399     else
00400     {
00401         variable_error (variable, _("no sweeps number"));
00402         goto exit_on_error;
00403     }
00404 #if DEBUG_VARIABLE
00405     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00406 #endif
00407 }
00408 if (algorithm == ALGORITHM_GENETIC)
00409 {
00410     // Obtaining bits representing each variable
00411     if (json_object_get_member (object, LABEL_NBITS))
00412     {
00413         variable->nbits
00414         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00415         if (error_code || !variable->nbits)
00416         {
00417             variable_error (variable, _("invalid bits number"));
00418             goto exit_on_error;
00419         }
00420     }
00421     else
00422     {
00423         variable_error (variable, _("no bits number"));
00424         goto exit_on_error;
00425     }
00426 }
00427 else if (nsteps)
00428 {
00429     variable->step = json_object_get_float (object,
LABEL_STEP, &error_code);
00430     if (error_code || variable->step < 0.)
00431     {
00432         variable_error (variable, _("bad step size"));
00433         goto exit_on_error;
00434     }
00435 }
00436 #if DEBUG_VARIABLE
00437 fprintf (stderr, "variable_open_json: end\n");
00438 #endif
00439 return 1;
00440 exit_on_error:
00441 variable_free (variable, INPUT_TYPE_JSON);
00442 #if DEBUG_VARIABLE
00443 fprintf (stderr, "variable_open_json: end\n");
00444 #endif
00445 return 0;
00446 }

```

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 135 of file [variable.c](#).

```

00137 {
00138     int error_code;
00139
00140     #if DEBUG_VARIABLE
00141     fprintf (stderr, "variable_open_xml: start\n");
00142     #endif
00143
00144     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145     if (!variable->name)
00146     {
00147         variable_error (variable, _("no name"));
00148         goto exit_on_error;
00149     }
00150     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00151     {
00152         variable->rangemin
00153         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00154                               &error_code);
00155         if (error_code)
00156         {
00157             variable_error (variable, _("bad minimum"));
00158             goto exit_on_error;
00159         }
00160     }
00161 }
  
```

```

00159     }
00160     variable->rangeminabs = xml_node_get_float_with_default
00161     (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00162     &error_code);
00163     if (error_code)
00164     {
00165         variable_error (variable, _("bad absolute minimum"));
00166         goto exit_on_error;
00167     }
00168     if (variable->rangemin < variable->rangeminabs)
00169     {
00170         variable_error (variable, _("minimum range not allowed"));
00171         goto exit_on_error;
00172     }
00173 }
00174 else
00175 {
00176     variable_error (variable, _("no minimum range"));
00177     goto exit_on_error;
00178 }
00179 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00180 {
00181     variable->rangemax
00182     = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00183     &error_code);
00184     if (error_code)
00185     {
00186         variable_error (variable, _("bad maximum"));
00187         goto exit_on_error;
00188     }
00189     variable->rangemaxabs = xml_node_get_float_with_default
00190     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00191     &error_code);
00192     if (error_code)
00193     {
00194         variable_error (variable, _("bad absolute maximum"));
00195         goto exit_on_error;
00196     }
00197     if (variable->rangemax > variable->rangemaxabs)
00198     {
00199         variable_error (variable, _("maximum range not allowed"));
00200         goto exit_on_error;
00201     }
00202     if (variable->rangemax < variable->rangemin)
00203     {
00204         variable_error (variable, _("bad range"));
00205         goto exit_on_error;
00206     }
00207 }
00208 else
00209 {
00210     variable_error (variable, _("no maximum range"));
00211     goto exit_on_error;
00212 }
00213 variable->precision
00214 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
00215     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 #if DEBUG_VARIABLE
00240     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00241 #endif
00242 }
00243 if (algorithm == ALGORITHM_GENETIC)
00244 {

```

```

00245     // Obtaining bits representing each variable
00246     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00247     {
00248         variable->nbits
00249         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00250                             &error_code);
00251         if (error_code || !variable->nbits)
00252         {
00253             variable_error (variable, _("invalid bits number"));
00254             goto exit_on_error;
00255         }
00256     }
00257     else
00258     {
00259         variable_error (variable, _("no bits number"));
00260         goto exit_on_error;
00261     }
00262 }
00263 else if (nsteps)
00264 {
00265     variable->step
00266     = xml_node_get_float (node, (const xmlChar *)
00267 LABEL_STEP, &error_code);
00267     if (error_code || variable->step < 0.)
00268     {
00269         variable_error (variable, _("bad step size"));
00270         goto exit_on_error;
00271     }
00272 }
00273
00274 #if DEBUG_VARIABLE
00275 fprintf (stderr, "variable_open_xml: end\n");
00276 #endif
00277 return 1;
00278 exit_on_error:
00279     variable_free (variable, INPUT_TYPE_XML);
00280 #if DEBUG_VARIABLE
00281 fprintf (stderr, "variable_open_xml: end\n");
00282 #endif
00283 return 0;
00284 }

```

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 2012-2017, 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
```

```

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, message);
00117     error_message = g_strdup (buffer);
00118 }
00119
00134 int
00135 variable_open_xml (Variable * variable, xmlNode * node,
00136                   unsigned int algorithm, unsigned int nsteps)
00137 {
00138     int error_code;
00139
00140 #if DEBUG_VARIABLE
00141     fprintf (stderr, "variable_open_xml: start\n");
00142 #endif
00143
00144     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145     if (!variable->name)
00146     {
00147         variable_error (variable, _("no name"));
00148         goto exit_on_error;
00149     }
00150     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00151     {
00152         variable->rangemin
00153             = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00154                                   &error_code);
00155         if (error_code)
00156         {
00157             variable_error (variable, _("bad minimum"));
00158             goto exit_on_error;
00159         }
00160         variable->rangeminabs = xml_node_get_float_with_default
00161             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00162             &error_code);
00163         if (error_code)
00164         {
00165             variable_error (variable, _("bad absolute minimum"));
00166             goto exit_on_error;
00167         }
00168         if (variable->rangemin < variable->rangeminabs)
00169         {
00170             variable_error (variable, _("minimum range not allowed"));
00171             goto exit_on_error;
00172         }
00173     }
00174     else
00175     {
00176         variable_error (variable, _("no minimum range"));
00177         goto exit_on_error;
00178     }
00179     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00180     {
00181         variable->rangemax
00182             = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00183                                   &error_code);
00184         if (error_code)
00185         {
00186             variable_error (variable, _("bad maximum"));

```

```

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

```

```

00272     }
00273
00274 #if DEBUG_VARIABLE
00275     fprintf (stderr, "variable_open_xml: end\n");
00276 #endif
00277     return 1;
00278 exit_on_error:
00279     variable_free (variable, INPUT_TYPE_XML);
00280 #if DEBUG_VARIABLE
00281     fprintf (stderr, "variable_open_xml: end\n");
00282 #endif
00283     return 0;
00284 }
00285
00300 int
00301 variable_open_json (Variable * variable, JsonNode * node,
00302                    unsigned int algorithm, unsigned int nsteps)
00303 {
00304     JsonObject *object;
00305     const char *label;
00306     int error_code;
00307 #if DEBUG_VARIABLE
00308     fprintf (stderr, "variable_open_json: start\n");
00309 #endif
00310     object = json_node_get_object (node);
00311     label = json_object_get_string_member (object, LABEL_NAME);
00312     if (!label)
00313     {
00314         variable_error (variable, _("no name"));
00315         goto exit_on_error;
00316     }
00317     variable->name = g_strdup (label);
00318     if (json_object_get_member (object, LABEL_MINIMUM))
00319     {
00320         variable->rangemin
00321             = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00322         if (error_code)
00323         {
00324             variable_error (variable, _("bad minimum"));
00325             goto exit_on_error;
00326         }
00327         variable->rangeminabs
00328             = json_object_get_float_with_default (object,
00329 LABEL_ABSOLUTE_MINIMUM,
00329                                         -G_MAXDOUBLE, &error_code);
00330         if (error_code)
00331         {
00332             variable_error (variable, _("bad absolute minimum"));
00333             goto exit_on_error;
00334         }
00335         if (variable->rangemin < variable->rangeminabs)
00336         {
00337             variable_error (variable, _("minimum range not allowed"));
00338             goto exit_on_error;
00339         }
00340     }
00341     else
00342     {
00343         variable_error (variable, _("no minimum range"));
00344         goto exit_on_error;
00345     }
00346     if (json_object_get_member (object, LABEL_MAXIMUM))
00347     {
00348         variable->rangemax
00349             = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00350         if (error_code)
00351         {
00352             variable_error (variable, _("bad maximum"));
00353             goto exit_on_error;
00354         }
00355         variable->rangemaxabs
00356             = json_object_get_float_with_default (object,
00357 LABEL_ABSOLUTE_MAXIMUM,
00357                                         G_MAXDOUBLE, &error_code);
00358         if (error_code)
00359         {
00360             variable_error (variable, _("bad absolute maximum"));
00361             goto exit_on_error;
00362         }
00363         if (variable->rangemax > variable->rangemaxabs)
00364         {
00365             variable_error (variable, _("maximum range not allowed"));
00366             goto exit_on_error;
00367         }
00368         if (variable->rangemax < variable->rangemin)
00369         {
00370             variable_error (variable, _("bad range"));

```



```

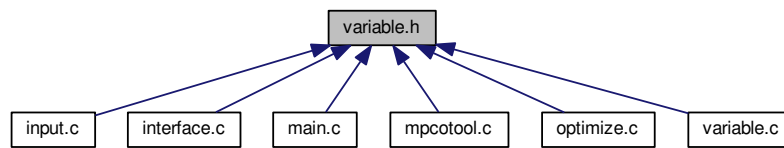
00371         goto exit_on_error;
00372     }
00373 }
00374 else
00375 {
00376     variable_error (variable, _("no maximum range"));
00377     goto exit_on_error;
00378 }
00379 variable->precision
00380 = json_object_get_uint_with_default (object,
LABEL_PRECISION,
00381                                     DEFAULT_PRECISION, &error_code);
00382 if (error_code || variable->precision >= NPRECISIONS)
00383 {
00384     variable_error (variable, _("bad precision"));
00385     goto exit_on_error;
00386 }
00387 if (algorithm == ALGORITHM_SWEEP)
00388 {
00389     if (json_object_get_member (object, LABEL_NSWEEPS))
00390     {
00391         variable->nsweeps
00392         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00393         if (error_code || !variable->nsweeps)
00394         {
00395             variable_error (variable, _("bad sweeps"));
00396             goto exit_on_error;
00397         }
00398     }
00399     else
00400     {
00401         variable_error (variable, _("no sweeps number"));
00402         goto exit_on_error;
00403     }
00404 #if DEBUG_VARIABLE
00405     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00406 #endif
00407 }
00408 if (algorithm == ALGORITHM_GENETIC)
00409 {
00410     // Obtaining bits representing each variable
00411     if (json_object_get_member (object, LABEL_NBITS))
00412     {
00413         variable->nbits
00414         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00415         if (error_code || !variable->nbits)
00416         {
00417             variable_error (variable, _("invalid bits number"));
00418             goto exit_on_error;
00419         }
00420     }
00421     else
00422     {
00423         variable_error (variable, _("no bits number"));
00424         goto exit_on_error;
00425     }
00426 }
00427 else if (nsteps)
00428 {
00429     variable->step = json_object_get_float (object,
LABEL_STEP, &error_code);
00430     if (error_code || variable->step < 0.)
00431     {
00432         variable_error (variable, _("bad step size"));
00433         goto exit_on_error;
00434     }
00435 }
00436
00437 #if DEBUG_VARIABLE
00438 fprintf (stderr, "variable_open_json: end\n");
00439 #endif
00440 return 1;
00441 exit_on_error:
00442 variable_free (variable, INPUT_TYPE_JSON);
00443 #if DEBUG_VARIABLE
00444 fprintf (stderr, "variable_open_json: end\n");
00445 #endif
00446 return 0;
00447 }

```

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, 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.

4.27.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2017, 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, message);
00117     error_message = g_strdup (buffer);
00118 }
```

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 301 of file [variable.c](#).

```

00303 {
00304     JsonObject *object;
00305     const char *label;
00306     int error_code;
00307     #if DEBUG_VARIABLE
00308         fprintf (stderr, "variable_open_json: start\n");
00309     #endif
00310     object = json_node_get_object (node);
00311     label = json_object_get_string_member (object, LABEL_NAME);
00312     if (!label)
00313     {
00314         variable_error (variable, _("no name"));
00315         goto exit_on_error;
00316     }
00317     variable->name = g_strdup (label);
00318     if (json_object_get_member (object, LABEL_MINIMUM))
00319     {
00320         variable->rangemin
```

```

00321         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00322     if (error_code)
00323     {
00324         variable_error (variable, _("bad minimum"));
00325         goto exit_on_error;
00326     }
00327     variable->rangeminabs
00328     = json_object_get_float_with_default (object,
00329     LABEL_ABSOLUTE_MINIMUM,
00329                                     -G_MAXDOUBLE, &error_code);
00330     if (error_code)
00331     {
00332         variable_error (variable, _("bad absolute minimum"));
00333         goto exit_on_error;
00334     }
00335     if (variable->rangemin < variable->rangeminabs)
00336     {
00337         variable_error (variable, _("minimum range not allowed"));
00338         goto exit_on_error;
00339     }
00340 }
00341 else
00342 {
00343     variable_error (variable, _("no minimum range"));
00344     goto exit_on_error;
00345 }
00346 if (json_object_get_member (object, LABEL_MAXIMUM))
00347 {
00348     variable->rangemax
00349     = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00350     if (error_code)
00351     {
00352         variable_error (variable, _("bad maximum"));
00353         goto exit_on_error;
00354     }
00355     variable->rangemaxabs
00356     = json_object_get_float_with_default (object,
00357     LABEL_ABSOLUTE_MAXIMUM,
00357                                     G_MAXDOUBLE, &error_code);
00358     if (error_code)
00359     {
00360         variable_error (variable, _("bad absolute maximum"));
00361         goto exit_on_error;
00362     }
00363     if (variable->rangemax > variable->rangemaxabs)
00364     {
00365         variable_error (variable, _("maximum range not allowed"));
00366         goto exit_on_error;
00367     }
00368     if (variable->rangemax < variable->rangemin)
00369     {
00370         variable_error (variable, _("bad range"));
00371         goto exit_on_error;
00372     }
00373 }
00374 else
00375 {
00376     variable_error (variable, _("no maximum range"));
00377     goto exit_on_error;
00378 }
00379 variable->precision
00380 = json_object_get_uint_with_default (object,
00381     LABEL_PRECISION,
00381     DEFAULT_PRECISION, &error_code);
00382 if (error_code || variable->precision >= NPRECISIONS)
00383 {
00384     variable_error (variable, _("bad precision"));
00385     goto exit_on_error;
00386 }
00387 if (algorithm == ALGORITHM_SWEEP)
00388 {
00389     if (json_object_get_member (object, LABEL_NSWEEPS))
00390     {
00391         variable->nsweeps
00392         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00393         if (error_code || !variable->nsweeps)
00394         {
00395             variable_error (variable, _("bad sweeps"));
00396             goto exit_on_error;
00397         }
00398     }
00399     else
00400     {
00401         variable_error (variable, _("no sweeps number"));
00402         goto exit_on_error;
00403     }
00404 #if DEBUG_VARIABLE

```

```

00405     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00406 #endif
00407 }
00408 if (algorithm == ALGORITHM_GENETIC)
00409 {
00410     // Obtaining bits representing each variable
00411     if (json_object_get_member (object, LABEL_NBITS))
00412     {
00413         variable->nbits
00414         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00415         if (error_code || !variable->nbits)
00416         {
00417             variable_error (variable, _("invalid bits number"));
00418             goto exit_on_error;
00419         }
00420     }
00421     else
00422     {
00423         variable_error (variable, _("no bits number"));
00424         goto exit_on_error;
00425     }
00426 }
00427 else if (nsteps)
00428 {
00429     variable->step = json_object_get_float (object,
00430 LABEL_STEP, &error_code);
00431     if (error_code || variable->step < 0.)
00432     {
00433         variable_error (variable, _("bad step size"));
00434         goto exit_on_error;
00435     }
00436 }
00437 #if DEBUG_VARIABLE
00438     fprintf (stderr, "variable_open_json: end\n");
00439 #endif
00440     return 1;
00441 exit_on_error:
00442     variable_free (variable, INPUT_TYPE_JSON);
00443 #if DEBUG_VARIABLE
00444     fprintf (stderr, "variable_open_json: end\n");
00445 #endif
00446     return 0;
00447 }

```

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 135 of file [variable.c](#).

```

00137 {
00138     int error_code;
00139
00140     #if DEBUG_VARIABLE
00141     fprintf (stderr, "variable_open_xml: start\n");
00142     #endif
00143
00144     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145     if (!variable->name)
00146     {
00147         variable_error (variable, _("no name"));
00148         goto exit_on_error;
00149     }
00150     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00151     {
00152         variable->rangemin
00153         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00154                               &error_code);
00155         if (error_code)
00156         {
00157             variable_error (variable, _("bad minimum"));
00158             goto exit_on_error;
00159         }
00160         variable->rangeminabs = xml_node_get_float_with_default
00161         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00162          &error_code);
00163         if (error_code)
00164         {
00165             variable_error (variable, _("bad absolute minimum"));
00166             goto exit_on_error;
00167         }
00168         if (variable->rangemin < variable->rangeminabs)
00169         {
00170             variable_error (variable, _("minimum range not allowed"));
00171             goto exit_on_error;
00172         }
00173     }
00174     else
00175     {
00176         variable_error (variable, _("no minimum range"));
00177         goto exit_on_error;
00178     }
00179     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00180     {
00181         variable->rangemax
00182         = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00183                               &error_code);
00184         if (error_code)
00185         {
00186             variable_error (variable, _("bad maximum"));
00187             goto exit_on_error;
00188         }
00189         variable->rangemaxabs = xml_node_get_float_with_default
00190         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00191          &error_code);
00192         if (error_code)
00193         {
00194             variable_error (variable, _("bad absolute maximum"));
00195             goto exit_on_error;
00196         }
00197         if (variable->rangemax > variable->rangemaxabs)
00198         {
00199             variable_error (variable, _("maximum range not allowed"));
00200             goto exit_on_error;

```



```

00201     }
00202     if (variable->rangemax < variable->rangemin)
00203     {
00204         variable_error (variable, _("bad range"));
00205         goto exit_on_error;
00206     }
00207 }
00208 else
00209 {
00210     variable_error (variable, _("no maximum range"));
00211     goto exit_on_error;
00212 }
00213 variable->precision
00214 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                DEFAULT_PRECISION, &error_code);
00215 if (error_code || variable->precision >= NPRECISIONS)
00216 {
00217     variable_error (variable, _("bad precision"));
00218     goto exit_on_error;
00219 }
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,
                                &error_code);
00227         if (error_code || !variable->nsweeps)
00228         {
00229             variable_error (variable, _("bad sweeps"));
00230             goto exit_on_error;
00231         }
00232     }
00233 }
00234 else
00235 {
00236     variable_error (variable, _("no sweeps number"));
00237     goto exit_on_error;
00238 }
00239 #if DEBUG_VARIABLE
00240     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00241 #endif
00242 }
00243 if (algorithm == ALGORITHM_GENETIC)
00244 {
00245     // Obtaining bits representing each variable
00246     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00247     {
00248         variable->nbits
00249         = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                &error_code);
00250         if (error_code || !variable->nbits)
00251         {
00252             variable_error (variable, _("invalid bits number"));
00253             goto exit_on_error;
00254         }
00255     }
00256 }
00257 else
00258 {
00259     variable_error (variable, _("no bits number"));
00260     goto exit_on_error;
00261 }
00262 }
00263 else if (nsteps)
00264 {
00265     variable->step
00266     = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
00267     if (error_code || variable->step < 0.)
00268     {
00269         variable_error (variable, _("bad step size"));
00270         goto exit_on_error;
00271     }
00272 }
00273 #if DEBUG_VARIABLE
00274     fprintf (stderr, "variable_open_xml: end\n");
00275 #endif
00276 return 1;
00277 exit_on_error:
00278     variable_free (variable, INPUT_TYPE_XML);
00279 #if DEBUG_VARIABLE
00280     fprintf (stderr, "variable_open_xml: end\n");
00281 #endif
00282 return 0;
00283 }
00284 }

```

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 2012-2017, 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, 273

config.h, 19
INPUT_TYPE, 22

cores_number
utils.c, 226
utils.h, 245

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
stencil, 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, 266

gtk_array_get_active
interface.h, 146
utils.c, 226
utils.h, 245

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, 68
DirectionMethod, 69
ErrorNorm, 69
input_error, 70
input_open, 70
input_open_json, 73
input_open_xml, 78

input_error
input.c, 43
input.h, 70

input_open
input.c, 44
input.h, 70

input_open_json
input.c, 45
input.h, 73

input_open_xml
input.c, 50
input.h, 78

input_save
interface.c, 88
interface.h, 147

input_save_direction_json
interface.c, 89

input_save_direction_xml
interface.c, 90

input_save_json
interface.c, 91

input_save_xml
interface.c, 93

interface.c, 85
input_save, 88
input_save_direction_json, 89
input_save_direction_xml, 90
input_save_json, 91
input_save_xml, 93
window_get_algorithm, 96
window_get_direction, 97
window_get_norm, 97
window_new, 98
window_read, 107

- window_save, 109
 - window_template_experiment, 111
- interface.h, 143
 - gtk_array_get_active, 146
 - input_save, 147
 - window_get_algorithm, 148
 - window_get_direction, 148
 - window_get_norm, 149
 - window_new, 150
 - window_read, 159
 - window_save, 161
 - window_template_experiment, 163
- json_object_get_float
 - utils.c, 227
 - utils.h, 246
- json_object_get_float_with_default
 - utils.c, 228
 - utils.h, 246
- json_object_get_int
 - utils.c, 229
 - utils.h, 247
- json_object_get_uint
 - utils.c, 229
 - utils.h, 248
- json_object_get_uint_with_default
 - utils.c, 230
 - utils.h, 249
- json_object_set_float
 - utils.c, 231
 - utils.h, 250
- json_object_set_int
 - utils.c, 231
 - utils.h, 250
- json_object_set_uint
 - utils.c, 232
 - utils.h, 250
- main.c, 166
- Optimize, 7
 - thread_direction, 10
- optimize.c, 168
 - optimize_best, 171
 - optimize_best_direction, 172
 - optimize_direction_sequential, 172
 - optimize_direction_thread, 173
 - optimize_estimate_direction_coordinates, 174
 - optimize_estimate_direction_random, 175
 - optimize_genetic_objective, 175
 - optimize_input, 176
 - optimize_merge, 178
 - optimize_norm_euclidian, 179
 - optimize_norm_maximum, 179
 - optimize_norm_p, 180
 - optimize_norm_taxicab, 181
 - optimize_parse, 182
 - optimize_save_variables, 184
 - optimize_step_direction, 184
 - optimize_thread, 186
- optimize.h, 204
 - optimize_best, 207
 - optimize_best_direction, 208
 - optimize_direction_sequential, 208
 - optimize_direction_thread, 209
 - optimize_estimate_direction_coordinates, 210
 - optimize_estimate_direction_random, 211
 - optimize_genetic_objective, 211
 - optimize_input, 212
 - optimize_merge, 214
 - optimize_norm_euclidian, 215
 - optimize_norm_maximum, 215
 - optimize_norm_p, 216
 - optimize_norm_taxicab, 217
 - optimize_parse, 218
 - optimize_save_variables, 220
 - optimize_step_direction, 220
 - optimize_thread, 222
- optimize_best
 - optimize.c, 171
 - optimize.h, 207
- optimize_best_direction
 - optimize.c, 172
 - optimize.h, 208
- optimize_direction_sequential
 - optimize.c, 172
 - optimize.h, 208
- optimize_direction_thread
 - optimize.c, 173
 - optimize.h, 209
- optimize_estimate_direction_coordinates
 - optimize.c, 174
 - optimize.h, 210
- optimize_estimate_direction_random
 - optimize.c, 175
 - optimize.h, 211
- optimize_genetic_objective
 - optimize.c, 175
 - optimize.h, 211
- optimize_input
 - optimize.c, 176
 - optimize.h, 212
- optimize_merge
 - optimize.c, 178
 - optimize.h, 214
- optimize_norm_euclidian
 - optimize.c, 179
 - optimize.h, 215
- optimize_norm_maximum
 - optimize.c, 179
 - optimize.h, 215
- optimize_norm_p
 - optimize.c, 180
 - optimize.h, 216
- optimize_norm_taxicab
 - optimize.c, 181
 - optimize.h, 217

- optimize_parse
 - optimize.c, [182](#)
 - optimize.h, [218](#)
- optimize_save_variables
 - optimize.c, [184](#)
 - optimize.h, [220](#)
- optimize_step_direction
 - optimize.c, [184](#)
 - optimize.h, [220](#)
- optimize_thread
 - optimize.c, [186](#)
 - optimize.h, [222](#)
- Options, [11](#)
- ParallelData, [11](#)
- precision
 - variable.c, [266](#)
- Running, [12](#)
- show_error
 - utils.c, [232](#)
 - utils.h, [251](#)
- show_message
 - utils.c, [233](#)
 - utils.h, [252](#)
- stencil
 - experiment.c, [31](#)
- thread_direction
 - Optimize, [10](#)
- utils.c, [224](#)
 - cores_number, [226](#)
 - gtk_array_get_active, [226](#)
 - json_object_get_float, [227](#)
 - json_object_get_float_with_default, [228](#)
 - json_object_get_int, [229](#)
 - json_object_get_uint, [229](#)
 - json_object_get_uint_with_default, [230](#)
 - json_object_set_float, [231](#)
 - json_object_set_int, [231](#)
 - json_object_set_uint, [232](#)
 - show_error, [232](#)
 - show_message, [233](#)
 - xml_node_get_float, [234](#)
 - xml_node_get_float_with_default, [234](#)
 - xml_node_get_int, [235](#)
 - xml_node_get_uint, [236](#)
 - xml_node_get_uint_with_default, [237](#)
 - xml_node_set_float, [238](#)
 - xml_node_set_int, [238](#)
 - xml_node_set_uint, [238](#)
- utils.h, [243](#)
 - cores_number, [245](#)
 - gtk_array_get_active, [245](#)
 - json_object_get_float, [246](#)
 - json_object_get_float_with_default, [246](#)
 - json_object_get_int, [247](#)
 - json_object_get_uint, [248](#)
 - json_object_get_uint_with_default, [249](#)
 - json_object_set_float, [250](#)
 - json_object_set_int, [250](#)
 - json_object_set_uint, [250](#)
 - show_error, [251](#)
 - show_message, [252](#)
 - xml_node_get_float, [252](#)
 - xml_node_get_float_with_default, [253](#)
 - xml_node_get_int, [254](#)
 - xml_node_get_uint, [254](#)
 - xml_node_get_uint_with_default, [255](#)
 - xml_node_set_float, [256](#)
 - xml_node_set_int, [256](#)
 - xml_node_set_uint, [257](#)
- Variable, [12](#)
- variable.c, [259](#)
 - format, [266](#)
 - precision, [266](#)
 - variable_error, [260](#)
 - variable_free, [260](#)
 - variable_new, [261](#)
 - variable_open_json, [261](#)
 - variable_open_xml, [264](#)
- variable.h, [271](#)
 - Algorithm, [273](#)
 - variable_error, [273](#)
 - variable_free, [274](#)
 - variable_new, [274](#)
 - variable_open_json, [275](#)
 - variable_open_xml, [277](#)
- variable_error
 - variable.c, [260](#)
 - variable.h, [273](#)
- variable_free
 - variable.c, [260](#)
 - variable.h, [274](#)
- variable_new
 - variable.c, [261](#)
 - variable.h, [274](#)
- variable_open_json
 - variable.c, [261](#)
 - variable.h, [275](#)
- variable_open_xml
 - variable.c, [264](#)
 - variable.h, [277](#)
- Window, [13](#)
- window_get_algorithm
 - interface.c, [96](#)
 - interface.h, [148](#)
- window_get_direction
 - interface.c, [97](#)
 - interface.h, [148](#)
- window_get_norm
 - interface.c, [97](#)
 - interface.h, [149](#)
- window_new

- interface.c, [98](#)
 - interface.h, [150](#)
- window_read
 - interface.c, [107](#)
 - interface.h, [159](#)
- window_save
 - interface.c, [109](#)
 - interface.h, [161](#)
- window_template_experiment
 - interface.c, [111](#)
 - interface.h, [163](#)
- xml_node_get_float
 - utils.c, [234](#)
 - utils.h, [252](#)
- xml_node_get_float_with_default
 - utils.c, [234](#)
 - utils.h, [253](#)
- xml_node_get_int
 - utils.c, [235](#)
 - utils.h, [254](#)
- xml_node_get_uint
 - utils.c, [236](#)
 - utils.h, [254](#)
- xml_node_get_uint_with_default
 - utils.c, [237](#)
 - utils.h, [255](#)
- xml_node_set_float
 - utils.c, [238](#)
 - utils.h, [256](#)
- xml_node_set_int
 - utils.c, [238](#)
 - utils.h, [256](#)
- xml_node_set_uint
 - utils.c, [238](#)
 - utils.h, [257](#)