

MPCOTool

2.1.1

Generated by Doxygen 1.8.9.1

Thu Jan 28 2016 15:22:31



# Contents

<b>1</b>	<b>MPCOTool</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>9</b>
2.1	Data Structures . . . . .	9
<b>3</b>	<b>File Index</b>	<b>11</b>
3.1	File List . . . . .	11
<b>4</b>	<b>Data Structure Documentation</b>	<b>13</b>
4.1	Experiment Struct Reference . . . . .	13
4.1.1	Detailed Description . . . . .	13
4.2	Input Struct Reference . . . . .	13
4.2.1	Detailed Description . . . . .	15
4.3	Optimize Struct Reference . . . . .	15
4.3.1	Detailed Description . . . . .	17
4.3.2	Field Documentation . . . . .	18
4.3.2.1	thread_direction . . . . .	18
4.4	Options Struct Reference . . . . .	18
4.4.1	Detailed Description . . . . .	18
4.5	ParallelData Struct Reference . . . . .	18
4.5.1	Detailed Description . . . . .	19
4.6	Running Struct Reference . . . . .	19
4.6.1	Detailed Description . . . . .	19
4.7	Variable Struct Reference . . . . .	19
4.7.1	Detailed Description . . . . .	20
4.8	Window Struct Reference . . . . .	20
4.8.1	Detailed Description . . . . .	25
<b>5</b>	<b>File Documentation</b>	<b>27</b>
5.1	config.h File Reference . . . . .	27
5.1.1	Detailed Description . . . . .	30
5.2	config.h . . . . .	30
5.3	experiment.c File Reference . . . . .	31

5.3.1	Detailed Description	32
5.3.2	Function Documentation	32
5.3.2.1	experiment_error	32
5.3.2.2	experiment_free	33
5.3.2.3	experiment_new	34
5.3.2.4	experiment_open	34
5.3.3	Variable Documentation	36
5.3.3.1	template	36
5.4	experiment.c	36
5.5	experiment.h File Reference	38
5.5.1	Detailed Description	39
5.5.2	Function Documentation	39
5.5.2.1	experiment_error	39
5.5.2.2	experiment_free	40
5.5.2.3	experiment_new	40
5.5.2.4	experiment_open	40
5.6	experiment.h	42
5.7	input.c File Reference	43
5.7.1	Detailed Description	44
5.7.2	Function Documentation	44
5.7.2.1	input_error	44
5.7.2.2	input_open	44
5.8	input.c	50
5.9	input.h File Reference	56
5.9.1	Detailed Description	57
5.9.2	Enumeration Type Documentation	58
5.9.2.1	DirectionMethod	58
5.9.2.2	ErrorNorm	58
5.9.3	Function Documentation	58
5.9.3.1	input_error	58
5.9.3.2	input_open	58
5.10	input.h	64
5.11	interface.c File Reference	65
5.11.1	Detailed Description	68
5.11.2	Function Documentation	68
5.11.2.1	input_save	68
5.11.2.2	input_save_direction	70
5.11.2.3	window_get_algorithm	71
5.11.2.4	window_get_direction	72
5.11.2.5	window_get_norm	72

5.11.2.6	<a href="#">window_read</a>	73
5.11.2.7	<a href="#">window_save</a>	75
5.11.2.8	<a href="#">window_template_experiment</a>	77
5.12	<a href="#">interface.c</a>	77
5.13	<a href="#">interface.h File Reference</a>	105
5.13.1	<a href="#">Detailed Description</a>	107
5.13.2	<a href="#">Function Documentation</a>	107
5.13.2.1	<a href="#">gtk_array_get_active</a>	107
5.13.2.2	<a href="#">input_save</a>	108
5.13.2.3	<a href="#">window_get_algorithm</a>	110
5.13.2.4	<a href="#">window_get_direction</a>	111
5.13.2.5	<a href="#">window_get_norm</a>	111
5.13.2.6	<a href="#">window_read</a>	112
5.13.2.7	<a href="#">window_save</a>	114
5.13.2.8	<a href="#">window_template_experiment</a>	116
5.14	<a href="#">interface.h</a>	116
5.15	<a href="#">main.c File Reference</a>	119
5.15.1	<a href="#">Detailed Description</a>	120
5.15.2	<a href="#">Function Documentation</a>	120
5.15.2.1	<a href="#">main</a>	120
5.16	<a href="#">main.c</a>	123
5.17	<a href="#">optimize.c File Reference</a>	126
5.17.1	<a href="#">Detailed Description</a>	128
5.17.2	<a href="#">Function Documentation</a>	129
5.17.2.1	<a href="#">optimize_best</a>	129
5.17.2.2	<a href="#">optimize_best_direction</a>	130
5.17.2.3	<a href="#">optimize_direction_sequential</a>	131
5.17.2.4	<a href="#">optimize_direction_thread</a>	132
5.17.2.5	<a href="#">optimize_estimate_direction_coordinates</a>	133
5.17.2.6	<a href="#">optimize_estimate_direction_random</a>	134
5.17.2.7	<a href="#">optimize_genetic_objective</a>	134
5.17.2.8	<a href="#">optimize_input</a>	135
5.17.2.9	<a href="#">optimize_merge</a>	136
5.17.2.10	<a href="#">optimize_norm_euclidian</a>	137
5.17.2.11	<a href="#">optimize_norm_maximum</a>	138
5.17.2.12	<a href="#">optimize_norm_p</a>	139
5.17.2.13	<a href="#">optimize_norm_taxicab</a>	140
5.17.2.14	<a href="#">optimize_parse</a>	140
5.17.2.15	<a href="#">optimize_save_variables</a>	142
5.17.2.16	<a href="#">optimize_step_direction</a>	143

5.17.2.17 optimize_thread . . . . .	144
5.18 optimize.c . . . . .	145
5.19 optimize.h File Reference . . . . .	162
5.19.1 Detailed Description . . . . .	165
5.19.2 Function Documentation . . . . .	165
5.19.2.1 optimize_best . . . . .	165
5.19.2.2 optimize_best_direction . . . . .	166
5.19.2.3 optimize_direction_thread . . . . .	166
5.19.2.4 optimize_estimate_direction_coordinates . . . . .	167
5.19.2.5 optimize_estimate_direction_random . . . . .	168
5.19.2.6 optimize_genetic_objective . . . . .	169
5.19.2.7 optimize_input . . . . .	169
5.19.2.8 optimize_merge . . . . .	171
5.19.2.9 optimize_norm_euclidian . . . . .	172
5.19.2.10 optimize_norm_maximum . . . . .	173
5.19.2.11 optimize_norm_p . . . . .	174
5.19.2.12 optimize_norm_taxicab . . . . .	175
5.19.2.13 optimize_parse . . . . .	176
5.19.2.14 optimize_save_variables . . . . .	178
5.19.2.15 optimize_step_direction . . . . .	178
5.19.2.16 optimize_thread . . . . .	179
5.20 optimize.h . . . . .	180
5.21 utils.c File Reference . . . . .	182
5.21.1 Detailed Description . . . . .	184
5.21.2 Function Documentation . . . . .	184
5.21.2.1 cores_number . . . . .	184
5.21.2.2 gtk_array_get_active . . . . .	184
5.21.2.3 show_error . . . . .	184
5.21.2.4 show_message . . . . .	185
5.21.2.5 xml_node_get_float . . . . .	185
5.21.2.6 xml_node_get_float_with_default . . . . .	186
5.21.2.7 xml_node_get_int . . . . .	187
5.21.2.8 xml_node_get_uint . . . . .	187
5.21.2.9 xml_node_get_uint_with_default . . . . .	188
5.21.2.10 xml_node_set_float . . . . .	188
5.21.2.11 xml_node_set_int . . . . .	189
5.21.2.12 xml_node_set_uint . . . . .	189
5.22 utils.c . . . . .	189
5.23 utils.h File Reference . . . . .	192
5.23.1 Detailed Description . . . . .	193

5.23.2	Function Documentation	193
5.23.2.1	cores_number	193
5.23.2.2	gtk_array_get_active	194
5.23.2.3	show_error	194
5.23.2.4	show_message	195
5.23.2.5	xml_node_get_float	196
5.23.2.6	xml_node_get_float_with_default	196
5.23.2.7	xml_node_get_int	197
5.23.2.8	xml_node_get_uint	198
5.23.2.9	xml_node_get_uint_with_default	198
5.23.2.10	xml_node_set_float	199
5.23.2.11	xml_node_set_int	199
5.23.2.12	xml_node_set_uint	199
5.24	utils.h	200
5.25	variable.c File Reference	201
5.25.1	Detailed Description	202
5.25.2	Function Documentation	202
5.25.2.1	variable_error	202
5.25.2.2	variable_free	202
5.25.2.3	variable_new	202
5.25.2.4	variable_open	203
5.25.3	Variable Documentation	205
5.25.3.1	format	205
5.25.3.2	precision	206
5.26	variable.c	206
5.27	variable.h File Reference	209
5.27.1	Detailed Description	210
5.27.2	Enumeration Type Documentation	210
5.27.2.1	Algorithm	210
5.27.3	Function Documentation	210
5.27.3.1	variable_error	210
5.27.3.2	variable_free	210
5.27.3.3	variable_new	211
5.27.3.4	variable_open	211
5.28	variable.h	214
	<b>Index</b>	<b>217</b>





# Chapter 1

## MPCOTool

The Multi-Purposes Calibration and Optimization Tool. A software to perform calibrations or optimizations of empirical parameters.

### VERSIONS

- 2.2.0: Stable and recommended version.
- 2.3.1: Developing version to do new features.

### AUTHORS

- Javier Burguete Tolosa ([jburguete@eead.csic.es](mailto:jburguete@eead.csic.es))
- Borja Latorre Garcés ([borja.latorre@csic.es](mailto:borja.latorre@csic.es))

### TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

- `gcc` or `clang` (to compile the source code)
- `make` (to build the executable file)
- `autoconf` (to generate the Makefile in different operative systems)
- `automake` (to check the operative system)
- `pkg-config` (to find the libraries to compile)
- `gsl` (to generate random numbers)
- `libxml` (to deal with XML files)
- `glib` (extended utilities of C to work with data, lists, mapped files, regular expressions, using multicores in shared memory machines, ...)
- `genetic` (genetic algorithm)

### OPTIONAL TOOLS AND LIBRARIES

- `gettext` (to work with different locales)
- `gtk+` (to create the interactive GUI tool)
- `openmpi` or `mpich` (to run in parallelized tasks on multiple computers)

- `doxygen` (standard comments format to generate documentation)
- `latex` (to build the PDF manuals)

## FILES

The source code has to have the following files:

- 2.2.0/configure.ac: configure generator.
- 2.2.0/Makefile.in: Makefile generator.
- 2.2.0/config.h.in: config header generator.
- 2.2.0/mpcotool.c: main source code.
- 2.2.0/mpcotool.h: main header code.
- 2.2.0/interface.h: interface header code.
- 2.2.0/build: script to build all.
- 2.2.0/logo.png: logo figure.
- 2.2.0/Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- [README.md](#): this file.
- tests/testX/\*: several tests to check the program working.
- locales/\*/LC\_MESSAGES/mpcotool.po: translation files.
- manuals/\*.eps: manual figures in EPS format.
- manuals/\*.png: manual figures in PNG format.
- manuals/\*.tex: documentation source files.
- applications/\*/\*: several practical application cases.
- check\_errors/\*.xml: several mistaken files to check error handling.

## BUILDING INSTRUCTIONS

This software has been built and tested in the following operative systems. Probably, it can be built in other systems, distributions, or versions but it has not been tested.

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

Dyson Illumos

FreeBSD 10.2

Linux Mint DE 2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 12, 14, and 15

1. Download the latest `genetic` doing on a terminal:

---

```
$ git clone https://github.com/jburguete/genetic.git
```

2. Download this repository:

```
$ git clone https://github.com/jburguete/mpcotool.git
```

3. Link the latest genetic version to genetic:

```
$ cd mpcotool/2.2.0
$ ln -s ../../genetic/1.0.0 genetic
```

4. Build doing on a terminal:

```
$ ./build
```

## OpenBSD 5.8

1. Select adequate versions:

```
$ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15
```

2. Then, in a terminal, follow steps 1 to 4 of the previous Debian 8 section.

## Microsoft Windows 7 (with MSYS2)

### Microsoft Windows 8.1 (with MSYS2)

1. Install **MSYS2** and the required libraries and utilities. You can follow detailed instructions in [install-unix](#)
2. Then, in a MSYS2 terminal, follow steps 1 to 4 of the previous Debian 8 section.
3. Optional Windows binary package can be built doing in the terminal:

```
$ make windist
```

## Fedora Linux 23

1. In order to use OpenMPI compilation do in a terminal (in 64 bits version):

```
$ export PATH=$PATH:/usr/lib64/openmpi/bin
```

2. Then, follow steps 1 to 4 of the previous Debian 8 section.

## MAKING MANUALS INSTRUCTIONS

On UNIX type systems you need **texlive** installed. On Windows systems you need **MiKTeX**. In order to compile the manuals you can type on a terminal:

```
$ make manuals
```

## MAKING TESTS INSTRUCTIONS

In order to build the tests follow the next instructions:

1. Link some tests that needs genetic library doing in a terminal (assuming that you are in the directory mpcotool/2.2.0):

```
$ cd ../tests/test2
$ ln -s ../../genetic/1.0.0 genetic
$ cd ../test3
$ ln -s ../../genetic/1.0.0 genetic
$ cd ../test4
$ ln -s ../../genetic/1.0.0 genetic
```

2. Build all tests doing in the same terminal:

```
$ cd ../2.2.0
$ make tests
```

## USER INSTRUCTIONS

- Command line in sequential mode:

```
$ ./mpcotoolbin [-nthreads X] input_file.xml
```

- Command line in parallelized mode (where X is the number of threads to open in every node):

```
$ mpirun [MPI options] ./mpcotoolbin [-nthreads X] input_file.xml
```

- The syntax of the simulator has to be:

```
$ ./simulator_name input_file_1 [input_file_2] [input_file_3] [input_file_4] output_file
```

- The syntax of the program to evaluate the objective function has to be (where the first data in the results file has to be the objective function value):

```
$ ./evaluator_name simulated_file data_file results_file
```

- On UNIX type systems the GUI application can be open doing on a terminal:

```
$ ./mpcotool
```

## INPUT FILE FORMAT

The format of the main input file is as:

```
“<?xml version="1.0"?> <calibrate simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm_
_type" nsimulations="simulations_number" niterations="iterations_number" tolerance="tolerance_value" nbest="best_
_number" npopulation="population_number" ngenerations="generations_number" mutation="mutation_
ratio" reproduction="reproduction_ratio" adaptation="adaptation_ratio" gradient_type="gradient_method_type"
nsteps="steps_number" relaxation="relaxation_paramter" nestimates="estimates_number" seed="random_
seed" result="result_file" variables="variables_file"> <experiment name="data_file_1" template1="template_1_1"
template2="template_1_2" ... weight="weight_1"/> ... <experiment name="data_file_N" template1="template_
_N_1" template2="template_N_2" ... weight="weight_N"/> <variable name="variable_1" minimum="min_value"
maximum="max_value" precision="precision_digits" sweeps="sweeps_number" nbits="bits_number" step="step_
size"> ... <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_
digits" sweeps="sweeps_number" nbits="bits_number" step="step_size"> </calibrate> “
```

with:

- **simulator**: simulator executable file name.
- **evaluator**: Optional. When needed is the evaluator executable file name.
- **seed**: Optional. Seed of the pseudo-random numbers generator (default value is 7007).
- **result**: Optional. It is the name of the optime result file (default name is "result").
- **variables**: Optional. It is the name of all simulated variables file (default name is "variables").

- **precision:** Optional, defined for each variable. Number of precision digits to evaluate the variable. 0 apply for integer numbers (default value is 14).
- **weight** Optional, defined for each experiment. Multiplies the objective value obtained for each experiment in the final objective function value (default value is 1).

Implemented algorithms are:

- **sweep:** Sweep brute force algorithm. It requires for each variable:
  - *sweeps*: number of sweeps to generate for each variable in every experiment.
 The total number of simulations to run is:
 
$$(\text{number of experiments}) \times (\text{variable 1 number of sweeps}) \times \dots \times (\text{variable n number of sweeps}) \times (\text{number of iterations})$$
- **Monte-Carlo:** Monte-Carlo brute force algorithm. It requires on calibrate:
  - *nsimulations*: number of simulations to run in every experiment.
 The total number of simulations to run is:
 
$$(\text{number of experiments}) \times (\text{number of simulations}) \times (\text{number of iterations})$$
- Both brute force algorithms can be iterated to improve convergence by using the following parameters:
  - *nbest*: number of best simulations to calculate convergence interval on next iteration (default 1).
  - *tolerance*: tolerance parameter to increase convergence interval (default 0).
  - *niterations*: number of iterations (default 1).
 It multiplies the total number of simulations:
 
$$\times (\text{number of iterations})$$
- Moreover, both brute force algorithms can be coupled with a gradient based method by using:
  - *gradient\_type*: method to estimate the gradient. Two options are currently available:
    - \* *coordinates*: coordinates descent method.  
It increases the total number of simulations by:
 
$$(\text{number of experiments}) \times (\text{number of iterations}) \times (\text{number of steps}) \times 2 \times (\text{number of variables})$$
    - \* *random*: random method. It requires:
    - \* *nestimates*: number of random checks to estimate the gradient.  
It increases the total number of simulations by:
 
$$(\text{number of experiments}) \times (\text{number of iterations}) \times (\text{number of steps}) \times (\text{number of estimates})$$

Both methods require also:

- *nsteps*: number of steps to perform the gradient based method,
- *relaxation*: relaxation parameter,

and for each variable:

- *step*: initial step size for the gradient based method.

- **genetic:** Genetic algorithm. It requires the following parameters:
  - *npopulation*: number of population.
  - *ngenerations*: number of generations.
  - *mutation*: mutation ratio.
  - *reproduction*: reproduction ratio.
  - *adaptation*: adaptation ratio.

and for each variable:

- *nbits*: number of bits to encode each variable.

The total number of simulations to run is:

(number of experiments) x (npopulation) x [1 + (ngenerations - 1) x (mutation + reproduction + adaptation)]

## SOME EXAMPLES OF INPUT FILES

### Example 1

- The simulator program name is: *pivot*

- The syntax is:

```
$ ./pivot input_file output_file
```

- The program to evaluate the objective function is: *compare*

- The syntax is:

```
$ ./compare simulated_file data_file result_file
```

- The calibration is performed with a *sweep brute force algorithm*.

- The experimental data files are:

27-48.txt

42.txt

52.txt

100.txt

- Templates to get input files to simulator for each experiment are:

template1.js

template2.js

template3.js

template4.js

- The variables to calibrate, ranges, precision and sweeps number to perform are:

alpha1, [179.70, 180.20], 2, 5

alpha2, [179.30, 179.60], 2, 5

random, [0.00, 0.20], 2, 5

boot-time, [0.0, 3.0], 1, 5

- Then, the number of simulations to run is: 4x5x5x5x5=2500.

- The input file is:

```
“<?xml version="1.0"?> <calibrate simulator="pivot" evaluator="compare" algorithm="sweep"> <experiment
name="27-48.txt" template1="template1.js"> <experiment name="42.txt" template1="template2.js"> <experiment
name="52.txt" template1="template3.js"> <experiment name="100.txt" template1="template4.js"> <variable
name="alpha1" minimum="179.70" maximum="180.20" precision="2" nsweeps="5"> <variable name="alpha2"
minimum="179.30" maximum="179.60" precision="2" nsweeps="5"> <variable name="random" minimum="0.00"
maximum="0.20" precision="2" nsweeps="5"> <variable name="boot-time" minimum="0.0" maximum="3.0"
precision="1" nsweeps="5"> </calibrate> “
```

- A template file as *template1.js*:

```

“ { "towers" : [ { "length" : 50.11, "velocity" : 0.02738, "@variable1@" : @, "@variable2@" : @, "@variable3@" :
@, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.02824, "@variable1@" : @, "@variable2@" : @, "@vari-
able3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03008, "@variable1@" : @, "@variable2@" :
@, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03753, "@variable1@" : @, "@vari-
able2@" : @, "@variable3@" : @, "@variable4@" : @ } ], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step":
0.1, "active-percent" : 27.48 } “

```

- produces simulator input files to reproduce the experimental data file *27-48.txt* as:

```

“json { "towers" : [ { "length" : 50.11, "velocity" : 0.02738, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10,
"boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.02824, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10,
"boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03008, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10,
"boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03753, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10,
"boot-time" : 1.5 } ], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “

```





## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">Experiment</a>	Struct to define the experiment data . . . . .	13
<a href="#">Input</a>	Struct to define the optimization input file . . . . .	13
<a href="#">Optimize</a>	Struct to define the optimization ation data . . . . .	15
<a href="#">Options</a>	Struct to define the options dialog . . . . .	18
<a href="#">ParallelData</a>	Struct to pass to the GThreads parallelized function . . . . .	18
<a href="#">Running</a>	Struct to define the running dialog . . . . .	19
<a href="#">Variable</a>	Struct to define the variable data . . . . .	19
<a href="#">Window</a>	Struct to define the main window . . . . .	20



## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">config.h</a>	Configuration header file . . . . .	27
<a href="#">experiment.c</a>	Source file to define the experiment data . . . . .	31
<a href="#">experiment.h</a>	Header file to define the experiment data . . . . .	38
<a href="#">input.c</a>	Source file to define the input functions . . . . .	43
<a href="#">input.h</a>	Header file to define the input functions . . . . .	56
<a href="#">interface.c</a>	Source file to define the graphical interface functions . . . . .	65
<a href="#">interface.h</a>	Header file to define the graphical interface functions . . . . .	105
<a href="#">main.c</a>	Main source file . . . . .	119
<a href="#">optimize.c</a>	Source file to define the optimization functions . . . . .	126
<a href="#">optimize.h</a>	Header file to define the optimization functions . . . . .	162
<a href="#">utils.c</a>	Source file to define some useful functions . . . . .	182
<a href="#">utils.h</a>	Header file to define some useful functions . . . . .	192
<a href="#">variable.c</a>	Source file to define the variable data . . . . .	201
<a href="#">variable.h</a>	Header file to define the variable data . . . . .	209



## Chapter 4

# Data Structure Documentation

### 4.1 Experiment Struct Reference

Struct to define the experiment data.

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

#### Data Fields

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

#### 4.1.1 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:

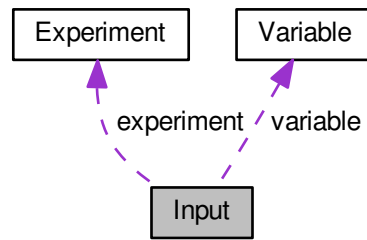
- [experiment.h](#)

### 4.2 Input Struct Reference

Struct to define the optimization input file.

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

Collaboration diagram for Input:



## Data Fields

- [Experiment](#) \* [experiment](#)  
*Array or experiments.*
- [Variable](#) \* [variable](#)  
*Array of variables.*
- char \* [result](#)  
*Name of the result file.*
- char \* [variables](#)  
*Name of the variables file.*
- char \* [simulator](#)  
*Name of the simulator program.*
- char \* [evaluator](#)  
*Name of the program to evaluate the objective function.*
- char \* [directory](#)  
*Working directory.*
- char \* [name](#)  
*[Input](#) data file name.*
- double [tolerance](#)  
*Algorithm tolerance.*
- double [mutation\\_ratio](#)  
*Mutation probability.*
- double [reproduction\\_ratio](#)  
*Reproduction probability.*
- double [adaptation\\_ratio](#)  
*Adaptation probability.*
- double [relaxation](#)  
*Relaxation parameter.*
- double [p](#)  
*Exponent of the P error norm.*
- double [thresold](#)  
*Thresold 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.*

#### 4.2.1 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](#)

## 4.3 Optimize Struct Reference

Struct to define the optimization ation data.

```
#include <optimize.h>
```

### 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](#)

*Thresold 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.*

#### 4.3.1 Detailed Description

Struct to define the optimization ation data.

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

### 4.3.2 Field Documentation

#### 4.3.2.1 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](#)

## 4.4 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.*

#### 4.4.1 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](#)

## 4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

```
#include <optimize.h>
```

## Data Fields

- unsigned int [thread](#)  
*Thread number.*

### 4.5.1 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](#)

## 4.6 Running Struct Reference

Struct to define the running dialog.

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

## Data Fields

- GtkWidget \* [dialog](#)  
*Main GtkWidget.*
- GtkWidget \* [label](#)  
*Label GtkWidget.*
- GtkWidget \* [spinner](#)  
*Animation GtkWidget.*
- GtkWidget \* [grid](#)  
*Grid GtkWidget.*

### 4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 66 of file [interface.h](#).

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

- [interface.h](#)

## 4.7 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.*

#### 4.7.1 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:

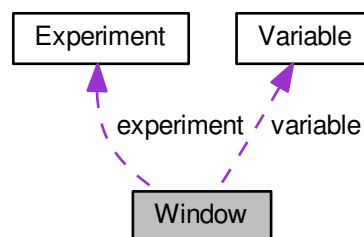
- [variable.h](#)

### 4.8 Window Struct Reference

Struct to define the main window.

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

Collaboration diagram for Window:



#### Data Fields

- GtkWidget \* [window](#)
- Main GtkWidget.*

- `GtkGrid * grid`  
*Main GtkGrid.*
- `GtkToolbar * bar_buttons`  
*GtkToolbar to store the main buttons.*
- `GtkToolButton * button_open`  
*Open GtkToolButton.*
- `GtkToolButton * button_save`  
*Save GtkToolButton.*
- `GtkToolButton * button_run`  
*Run GtkToolButton.*
- `GtkToolButton * button_options`  
*Options GtkToolButton.*
- `GtkToolButton * button_help`  
*Help GtkToolButton.*
- `GtkToolButton * button_about`  
*Help GtkToolButton.*
- `GtkToolButton * button_exit`  
*Exit GtkToolButton.*
- `GtkGrid * grid_files`  
*Files GtkGrid.*
- `GtkLabel * label_simulator`  
*Simulator program GtkLabel.*
- `GtkFileChooserButton * button_simulator`  
*Simulator program GtkFileChooserButton.*
- `GtkCheckButton * check_evaluator`  
*Evaluator program GtkCheckButton.*
- `GtkFileChooserButton * button_evaluator`  
*Evaluator program GtkFileChooserButton.*
- `GtkLabel * label_result`  
*Result file GtkLabel.*
- `GtkEntry * entry_result`  
*Result file GtkEntry.*
- `GtkLabel * label_variables`  
*Variables file GtkLabel.*
- `GtkEntry * entry_variables`  
*Variables file GtkEntry.*
- `GtkFrame * frame_norm`  
*GtkFrame to set the error norm.*
- `GtkGrid * grid_norm`  
*GtkGrid to set the error norm.*
- `GtkRadioButton * button_norm [NNORMS]`  
*Array of GtkButtons to set the error norm.*
- `GtkLabel * label_p`  
*GtkLabel to set the p parameter.*
- `GtkSpinButton * spin_p`  
*GtkSpinButton to set the p parameter.*
- `GtkScrolledWindow * scrolled_p`  
*GtkScrolledWindow to set the p parameter.*
- `GtkFrame * frame_algorithm`  
*GtkFrame to set the algorithm.*
- `GtkGrid * grid_algorithm`

- GtkGrid to set the algorithm.*
- GtkRadioButton \* [button\\_algorithm](#) [NALGORITHMS]  
*Array of 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\\_thresold](#)  
*GtkLabel to set the thresold.*
- GtkSpinButton \* [spin\\_thresold](#)  
*GtkSpinButton to set the thresold.*
- GtkScrolledWindow \* [scrolled\\_thresold](#)  
*GtkScrolledWindow to set the thresold.*
- 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.*

#### 4.8.1 Detailed Description

Struct to define the main window.

Definition at line 78 of file [interface.h](#).

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

- [interface.h](#)



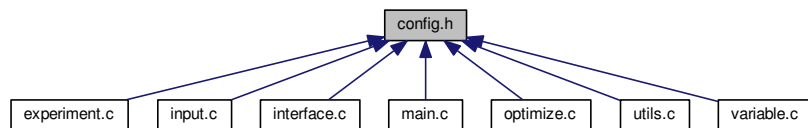
## Chapter 5

# File Documentation

### 5.1 config.h File Reference

Configuration header file.

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



### Macros

- `#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 `XML_ABSOLUTE_MINIMUM` (const xmlChar\*)"absolute\_minimum"  
*absolute minimum XML label.*
- #define `XML_ABSOLUTE_MAXIMUM` (const xmlChar\*)"absolute\_maximum"  
*absolute maximum XML label.*
- #define `XML_ADAPTATION` (const xmlChar\*)"adaptation"  
*adaption XML label.*
- #define `XML_ALGORITHM` (const xmlChar\*)"algorithm"  
*algorith XML label.*
- #define `XML_OPTIMIZE` (const xmlChar\*)"optimize"  
*optimize XML label.*
- #define `XML_COORDINATES` (const xmlChar\*)"coordinates"  
*coordinates XML label.*
- #define `XML_DIRECTION` (const xmlChar\*)"direction"  
*direction XML label.*
- #define `XML_EUCLIDIAN` (const xmlChar\*)"euclidian"  
*euclidian XML label.*
- #define `XML_EVALUATOR` (const xmlChar\*)"evaluator"  
*evaluator XML label.*
- #define `XML_EXPERIMENT` (const xmlChar\*)"experiment"  
*experiment XML label.*
- #define `XML_GENETIC` (const xmlChar\*)"genetic"  
*genetic XML label.*
- #define `XML_MINIMUM` (const xmlChar\*)"minimum"  
*minimum XML label.*
- #define `XML_MAXIMUM` (const xmlChar\*)"maximum"  
*maximum XML label.*
- #define `XML_MONTE_CARLO` (const xmlChar\*)"Monte-Carlo"  
*Monte-Carlo XML label.*
- #define `XML_MUTATION` (const xmlChar\*)"mutation"  
*mutation XML label.*
- #define `XML_NAME` (const xmlChar\*)"name"  
*name XML label.*
- #define `XML_NBEST` (const xmlChar\*)"nbest"  
*nbest XML label.*
- #define `XML_NBITS` (const xmlChar\*)"nbits"  
*nbits XML label.*
- #define `XML_NESTIMATES` (const xmlChar\*)"nestimates"  
*nestimates XML label.*
- #define `XML_NGENERATIONS` (const xmlChar\*)"ngenerations"  
*ngenerations XML label.*
- #define `XML_NITERATIONS` (const xmlChar\*)"niterations"  
*niterations XML label.*
- #define `XML_NORM` (const xmlChar\*)"norm"  
*norm XML label.*
- #define `XML_NPOPULATION` (const xmlChar\*)"npopulation"  
*npopulation XML label.*
- #define `XML_NSIMULATIONS` (const xmlChar\*)"nsimulations"  
*nsimulations XML label.*
- #define `XML_NSTEPS` (const xmlChar\*)"nsteps"  
*nsteps XML label.*
- #define `XML_NSWEEPS` (const xmlChar\*)"nsweps"

- *nsweeps XML label.*
- #define XML\_P (const xmlChar\*)"p"
- *p XML label.*
- #define XML\_PRECISION (const xmlChar\*)"precision"
- *precision XML label.*
- #define XML\_RANDOM (const xmlChar\*)"random"
- *random XML label.*
- #define XML\_RELAXATION (const xmlChar\*)"relaxation"
- *relaxation XML label.*
- #define XML\_REPRODUCTION (const xmlChar\*)"reproduction"
- *reproduction XML label.*
- #define XML\_RESULT (const xmlChar\*)"result"
- *result XML label.*
- #define XML\_SIMULATOR (const xmlChar\*)"simulator"
- *simulator XML label.*
- #define XML\_SEED (const xmlChar\*)"seed"
- *seed XML label.*
- #define XML\_STEP (const xmlChar\*)"step"
- *step XML label.*
- #define XML\_SWEEP (const xmlChar\*)"sweep"
- *sweep XML label.*
- #define XML\_TAXICAB (const xmlChar\*)"taxicab"
- *taxicab XML label.*
- #define XML\_TEMPLATE1 (const xmlChar\*)"template1"
- *template1 XML label.*
- #define XML\_TEMPLATE2 (const xmlChar\*)"template2"
- *template2 XML label.*
- #define XML\_TEMPLATE3 (const xmlChar\*)"template3"
- *template3 XML label.*
- #define XML\_TEMPLATE4 (const xmlChar\*)"template4"
- *template4 XML label.*
- #define XML\_TEMPLATE5 (const xmlChar\*)"template5"
- *template5 XML label.*
- #define XML\_TEMPLATE6 (const xmlChar\*)"template6"
- *template6 XML label.*
- #define XML\_TEMPLATE7 (const xmlChar\*)"template7"
- *template7 XML label.*
- #define XML\_TEMPLATE8 (const xmlChar\*)"template8"
- *template8 XML label.*
- #define XML\_THRESHOLD (const xmlChar\*)"threshold"
- *threshold XML label.*
- #define XML\_TOLERANCE (const xmlChar\*)"tolerance"
- *tolerance XML label.*
- #define XML\_VARIABLE (const xmlChar\*)"variable"
- *variable XML label.*
- #define XML\_VARIABLES (const xmlChar\*)"variables"
- *variables XML label.*
- #define XML\_WEIGHT (const xmlChar\*)"weight"
- *weight XML label.*

### 5.1.1 Detailed Description

Configuration header file.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

## 5.2 config.h

```

00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2016, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014     1. Redistributions of source code must retain the above copyright notice,
00015        this list of conditions and the following disclaimer.
00016
00017     2. Redistributions in binary form must reproduce the above copyright notice,
00018        this list of conditions and the following disclaimer in the
00019        documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00033 #ifndef CONFIG_H
00034 #define CONFIG_H 1
00035
00036 // Array sizes
00037
00038 #define MAX_NINPUTS 8
00039 #define NALGORITHMS 3
00040 #define NDIRECTIONS 2
00041 #define NNORMS 4
00042 #define NPRECISIONS 15
00043
00044 // Default choices
00045 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00046 #define DEFAULT_RANDOM_SEED 7007
00047 #define DEFAULT_RELAXATION 1.
00048
00049 // Interface labels
00050 #define LOCALE_DIR "locales"
00051 #define PROGRAM_INTERFACE "mpcotool"
00052
00053 // XML labels
00054 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00055 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00056 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00057 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00058 #define XML_OPTIMIZE (const xmlChar*)"optimize"

```

```

00073 #define XML_COORDINATES (const xmlChar*)"coordinates"
00075 #define XML_DIRECTION (const xmlChar*)"direction"
00077 #define XML_EUCLIDIAN (const xmlChar*)"euclidian"
00079 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00081 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00083 #define XML_GENETIC (const xmlChar*)"genetic"
00085 #define XML_MINIMUM (const xmlChar*)"minimum"
00086 #define XML_MAXIMUM (const xmlChar*)"maximum"
00087 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00088 #define XML_MUTATION (const xmlChar*)"mutation"
00090 #define XML_NAME (const xmlChar*)"name"
00091 #define XML_NBEST (const xmlChar*)"nbest"
00092 #define XML_NBITS (const xmlChar*)"nbits"
00093 #define XML_NESTIMATES (const xmlChar*)"nestimates"
00094 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00096 #define XML_NITERATIONS (const xmlChar*)"niterations"
00098 #define XML_NORM (const xmlChar*)"norm"
00100 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00101 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00103 #define XML_NSTEPS (const xmlChar*)"nsteps"
00105 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00106 #define XML_P (const xmlChar*)"p"
00107 #define XML_PRECISION (const xmlChar*)"precision"
00108 #define XML_RANDOM (const xmlChar*)"random"
00110 #define XML_RELAXATION (const xmlChar*)"relaxation"
00111 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00113 #define XML_RESULT (const xmlChar*)"result"
00115 #define XML_SIMULATOR (const xmlChar*)"simulator"
00116 #define XML_SEED (const xmlChar*)"seed"
00118 #define XML_STEP (const xmlChar*)"step"
00119 #define XML_SWEEP (const xmlChar*)"sweep"
00120 #define XML_TAXICAB (const xmlChar*)"taxicab"
00121 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00122 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00124 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00126 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00128 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00130 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00132 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00134 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00136 #define XML_THRESHOLD (const xmlChar*)"threshold"
00138 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00140 #define XML_VARIABLE (const xmlChar*)"variable"
00142 #define XML_VARIABLES (const xmlChar*)"variables"
00143 #define XML_WEIGHT (const xmlChar*)"weight"
00145
00146 #endif

```

## 5.3 experiment.c File Reference

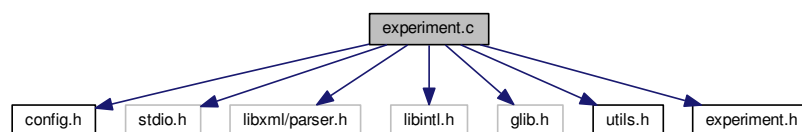
Source file to define the experiment data.

```

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

```

Include dependency graph for experiment.c:



## Macros

- `#define _GNU_SOURCE`
- `#define DEBUG 0`  
*Macro to debug.*

## Functions

- void `experiment_new` (`Experiment *experiment`)  
*Function to create a new `Experiment` struct.*
- void `experiment_free` (`Experiment *experiment`)  
*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` (`Experiment *experiment`, `xmlNode *node`, unsigned int `ninputs`)  
*Function to open the `Experiment` struct on a XML node.*

## Variables

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

### 5.3.1 Detailed Description

Source file to define the experiment data.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

Definition in file `experiment.c`.

### 5.3.2 Function Documentation

#### 5.3.2.1 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 109 of file `experiment.c`.

```

00110 {
00111     char buffer[64];
00112     if (!experiment->name)
00113         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00114     else
00115         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00116                 message);
00117     error_message = g_strdup (buffer);
00118 }
```



5.3.2.2 void experiment\_free ( Experiment \* *experiment* )

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

## Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
-------------------	------------------------------------

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

```

00086 {
00087     unsigned int i;
00088     #if DEBUG
00089         fprintf (stderr, "experiment_free: start\n");
00090     #endif
00091     for (i = 0; i < experiment->ninputs; ++i)
00092         xmlFree (experiment->template[i]);
00093     xmlFree (experiment->name);
00094     experiment->ninputs = 0;
00095     #if DEBUG
00096         fprintf (stderr, "experiment_free: end\n");
00097     #endif
00098 }
```

### 5.3.2.3 void experiment\_new ( Experiment \* *experiment* )

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

## Parameters

<i>experiment</i>	<a href="#">Experiment</a> struct.
-------------------	------------------------------------

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

```

00064 {
00065     unsigned int i;
00066     #if DEBUG
00067         fprintf (stderr, "experiment_new: start\n");
00068     #endif
00069     experiment->name = NULL;
00070     experiment->ninputs = 0;
00071     for (i = 0; i < MAX_NINPUTS; ++i)
00072         experiment->template[i] = NULL;
00073     #if DEBUG
00074         fprintf (stderr, "input_new: end\n");
00075     #endif
00076 }
```

### 5.3.2.4 int experiment\_open ( Experiment \* *experiment*, xmlNode \* *node*, unsigned int *ninputs* )

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

## Parameters

<i>experiment</i>	<a href="#">Experiment</a> 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 133 of file [experiment.c](#).

```

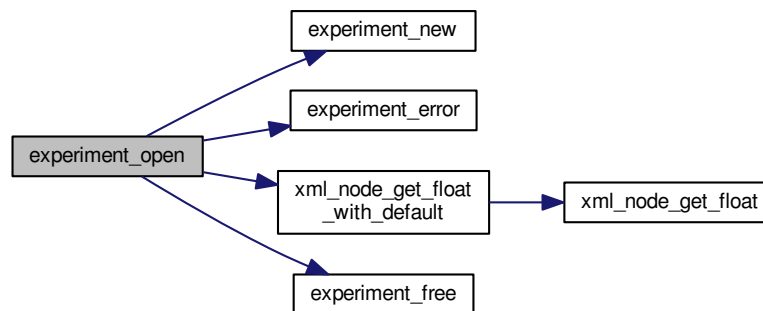
00134 {
00135     char buffer[64];
00136     int error_code;
00137     unsigned int i;
00138
00139     #if DEBUG
00140         fprintf (stderr, "experiment_open: start\n");
00141     #endif
00142 }
```

```

00143 // Resetting experiment data
00144 experiment_new (experiment);
00145
00146 // Reading the experimental data
00147 experiment->name = (char *) xmlGetProp (node, XML_NAME);
00148 if (!experiment->name)
00149 {
00150     experiment_error (experiment, gettext ("no data file name"));
00151     goto exit_on_error;
00152 }
00153 #if DEBUG
00154 fprintf (stderr, "experiment_open: name=%s\n", experiment->name);
00155 #endif
00156 experiment->weight
00157 = xml_node_get_float_with_default (node,
XML_WEIGHT, 1., &error_code);
00158 if (error_code)
00159 {
00160     experiment_error (experiment, gettext ("bad weight"));
00161     goto exit_on_error;
00162 }
00163 #if DEBUG
00164 fprintf (stderr, "experiment_open: weight=%lg\n", experiment->weight);
00165 #endif
00166 experiment->template[0] = (char *) xmlGetProp (node, template[0]);
00167 if (experiment->template[0])
00168 {
00169 #if DEBUG
00170     fprintf (stderr, "experiment_open: experiment=%s template1=%s\n",
00171             experiment->name, buffer2[0]);
00172 #endif
00173     ++experiment->ninputs;
00174 }
00175 else
00176 {
00177     experiment_error (experiment, gettext ("no template"));
00178     goto exit_on_error;
00179 }
00180 for (i = 1; i < MAX_NINPUTS; ++i)
00181 {
00182 #if DEBUG
00183     fprintf (stderr, "experiment_open: template%u\n", i + 1);
00184 #endif
00185     if (xmlHasProp (node, template[i]))
00186     {
00187         if (ninputs && ninputs <= i)
00188         {
00189             experiment_error (experiment, gettext ("bad templates number"));
00190             goto exit_on_error;
00191         }
00192         experiment->template[i] = (char *) xmlGetProp (node, template[i]);
00193 #if DEBUG
00194         fprintf (stderr, "experiment_open: experiment=%s template%u=%s\n",
00195                 experiment->name, experiment->name,
00196                 experiment->template[i]);
00197 #endif
00198         ++experiment->ninputs;
00199     }
00200     else if (ninputs && ninputs > i)
00201     {
00202         snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00203         experiment_error (experiment, buffer);
00204         goto exit_on_error;
00205     }
00206     else
00207         break;
00208 }
00209
00210 #if DEBUG
00211 fprintf (stderr, "experiment_open: end\n");
00212 #endif
00213 return 1;
00214
00215 exit_on_error:
00216 experiment_free (experiment);
00217 #if DEBUG
00218 fprintf (stderr, "experiment_open: end\n");
00219 #endif
00220 return 0;
00221 }

```

Here is the call graph for this function:



### 5.3.3 Variable Documentation

#### 5.3.3.1 `const xmlChar* template[MAX_NINPUTS]`

Initial value:

```

= {
    XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
    XML_TEMPLATE4,
    XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
    XML_TEMPLATE8
}

```

Array of `xmlChar` strings with template labels.

Definition at line 49 of file [experiment.c](#).

## 5.4 `experiment.c`

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */

```

```

00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include "utils.h"
00045 #include "experiment.h"
00046
00047 #define DEBUG 0
00048
00049 const xmlChar *template[MAX_NINPUTS] = {
00050     XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
00051     XML_TEMPLATE4,
00052     XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
00053     XML_TEMPLATE8
00054 };
00055
00062 void
00063 experiment_new (Experiment * experiment)
00064 {
00065     unsigned int i;
00066     #if DEBUG
00067         fprintf (stderr, "experiment_new: start\n");
00068     #endif
00069     experiment->name = NULL;
00070     experiment->ninputs = 0;
00071     for (i = 0; i < MAX_NINPUTS; ++i)
00072         experiment->template[i] = NULL;
00073     #if DEBUG
00074         fprintf (stderr, "input_new: end\n");
00075     #endif
00076 }
00077
00084 void
00085 experiment_free (Experiment * experiment)
00086 {
00087     unsigned int i;
00088     #if DEBUG
00089         fprintf (stderr, "experiment_free: start\n");
00090     #endif
00091     for (i = 0; i < experiment->ninputs; ++i)
00092         xmlFree (experiment->template[i]);
00093     xmlFree (experiment->name);
00094     experiment->ninputs = 0;
00095     #if DEBUG
00096         fprintf (stderr, "experiment_free: end\n");
00097     #endif
00098 }
00099
00108 void
00109 experiment_error (Experiment * experiment, char *message)
00110 {
00111     char buffer[64];
00112     if (!experiment->name)
00113         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00114     else
00115         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00116                 message);
00117     error_message = g_strdup (buffer);
00118 }
00119
00132 int
00133 experiment_open (Experiment * experiment, xmlNode * node, unsigned int ninputs)
00134 {
00135     char buffer[64];
00136     int error_code;
00137     unsigned int i;
00138
00139     #if DEBUG
00140         fprintf (stderr, "experiment_open: start\n");
00141     #endif
00142     // Resetting experiment data
00143     experiment_new (experiment);
00144     // Reading the experimental data
00145     experiment->name = (char *) xmlGetProp (node, XML_NAME);
00146     if (!experiment->name)
00147     {
00148         experiment_error (experiment, gettext ("no data file name"));
00149         goto exit_on_error;
00150     }
00151     #if DEBUG
00152         fprintf (stderr, "experiment_open: name=%s\n", experiment->name);
00153     #endif

```

```

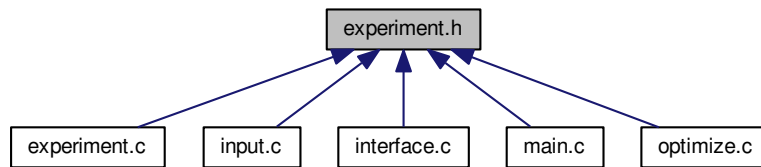
00155 #endif
00156     experiment->weight
00157     = xml_node_get_float_with_default (node,
XML_WEIGHT, 1., &error_code);
00158     if (error_code)
00159     {
00160         experiment_error (experiment, gettext ("bad weight"));
00161         goto exit_on_error;
00162     }
00163 #if DEBUG
00164     fprintf (stderr, "experiment_open: weight=%lg\n", experiment->weight);
00165 #endif
00166     experiment->template[0] = (char *) xmlGetProp (node, template[0]);
00167     if (experiment->template[0])
00168     {
00169 #if DEBUG
00170         fprintf (stderr, "experiment_open: experiment=%s template1=%s\n",
experiment->name, buffer2[0]);
00171 #endif
00172         ++experiment->ninputs;
00173     }
00174     else
00175     {
00176         experiment_error (experiment, gettext ("no template"));
00177         goto exit_on_error;
00178     }
00179     for (i = 1; i < MAX_NINPUTS; ++i)
00180     {
00181 #if DEBUG
00182         fprintf (stderr, "experiment_open: template%u\n", i + 1);
00183 #endif
00184         if (xmlHasProp (node, template[i]))
00185         {
00186             if (ninputs && ninputs <= i)
00187             {
00188                 experiment_error (experiment, gettext ("bad templates number"));
00189                 goto exit_on_error;
00190             }
00191             experiment->template[i] = (char *) xmlGetProp (node, template[i]);
00192 #if DEBUG
00193             fprintf (stderr, "experiment_open: experiment=%s template%u=%s\n",
experiment->name, experiment->nexperiments, experiment->template[i]);
00194 #endif
00195             ++experiment->ninputs;
00196         }
00197         else if (ninputs && ninputs > i)
00198         {
00199             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00200             experiment_error (experiment, buffer);
00201             goto exit_on_error;
00202         }
00203         else
00204             break;
00205     }
00206 #if DEBUG
00207     fprintf (stderr, "experiment_open: end\n");
00208 #endif
00209     return 1;
00210 }
00211 exit_on_error:
00212     experiment_free (experiment);
00213 #if DEBUG
00214     fprintf (stderr, "experiment_open: end\n");
00215 #endif
00216     return 0;
00217 }

```

## 5.5 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)  
*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](#) ([Experiment](#) \*experiment, xmlNode \*node, unsigned int ninputs)  
*Function to open the [Experiment](#) struct on a XML node.*

## Variables

- const xmlChar \* [template](#) [[MAX\\_NINPUTS](#)]  
*Array of xmlChar strings with template labels.*

### 5.5.1 Detailed Description

Header file to define the experiment data.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.5.2 Function Documentation

#### 5.5.2.1 void [experiment\\_error](#) ( [Experiment](#) \* *experiment*, char \* *message* )

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

## Parameters

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

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

```

00110 {
00111     char buffer[64];
00112     if (!experiment->name)
00113         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00114     else
00115         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00116                 message);
00117     error_message = g_strdup (buffer);
00118 }
```

### 5.5.2.2 void experiment\_free ( Experiment \* *experiment* )

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

## Parameters

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

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

```

00086 {
00087     unsigned int i;
00088     #if DEBUG
00089     fprintf (stderr, "experiment_free: start\n");
00090     #endif
00091     for (i = 0; i < experiment->ninputs; ++i)
00092         xmlFree (experiment->template[i]);
00093     xmlFree (experiment->name);
00094     experiment->ninputs = 0;
00095     #if DEBUG
00096     fprintf (stderr, "experiment_free: end\n");
00097     #endif
00098 }
```

### 5.5.2.3 void experiment\_new ( Experiment \* *experiment* )

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

## Parameters

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

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

```

00064 {
00065     unsigned int i;
00066     #if DEBUG
00067     fprintf (stderr, "experiment_new: start\n");
00068     #endif
00069     experiment->name = NULL;
00070     experiment->ninputs = 0;
00071     for (i = 0; i < MAX_NINPUTS; ++i)
00072         experiment->template[i] = NULL;
00073     #if DEBUG
00074     fprintf (stderr, "input_new: end\n");
00075     #endif
00076 }
```

### 5.5.2.4 int experiment\_open ( 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 133 of file [experiment.c](#).

```

00134 {
00135     char buffer[64];
00136     int error_code;
00137     unsigned int i;
00138
00139     #if DEBUG
00140         fprintf (stderr, "experiment_open: start\n");
00141     #endif
00142
00143     // Resetting experiment data
00144     experiment_new (experiment);
00145
00146     // Reading the experimental data
00147     experiment->name = (char *) xmlGetProp (node, XML_NAME);
00148     if (!experiment->name)
00149     {
00150         experiment_error (experiment, gettext ("no data file name"));
00151         goto exit_on_error;
00152     }
00153     #if DEBUG
00154         fprintf (stderr, "experiment_open: name=%s\n", experiment->name);
00155     #endif
00156     experiment->weight
00157         = xml_node_get_float_with_default (node,
00158         XML_WEIGHT, 1., &error_code);
00159     if (error_code)
00160     {
00161         experiment_error (experiment, gettext ("bad weight"));
00162         goto exit_on_error;
00163     }
00164     #if DEBUG
00165         fprintf (stderr, "experiment_open: weight=%lg\n", experiment->weight);
00166     #endif
00167     experiment->template[0] = (char *) xmlGetProp (node, template[0]);
00168     if (experiment->template[0])
00169     {
00170         #if DEBUG
00171             fprintf (stderr, "experiment_open: experiment=%s template1=%s\n",
00172             experiment->name, buffer2[0]);
00173         #endif
00174         ++experiment->ninputs;
00175     }
00176     else
00177     {
00178         experiment_error (experiment, gettext ("no template"));
00179         goto exit_on_error;
00180     }
00181     for (i = 1; i < MAX_NINPUTS; ++i)
00182     {
00183         #if DEBUG
00184             fprintf (stderr, "experiment_open: template%u\n", i + 1);
00185         #endif
00186         if (xmlHasProp (node, template[i]))
00187         {
00188             if (ninputs && ninputs <= i)
00189             {
00190                 experiment_error (experiment, gettext ("bad templates number"));
00191                 goto exit_on_error;
00192             }
00193             experiment->template[i] = (char *) xmlGetProp (node, template[i]);
00194             #if DEBUG
00195                 fprintf (stderr, "experiment_open: experiment=%s template%u=%s\n",
00196                 experiment->name, experiment->template[i]);
00197             #endif
00198             ++experiment->ninputs;
00199         }
00200         else if (ninputs && ninputs > i)
00201         {
00202             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);

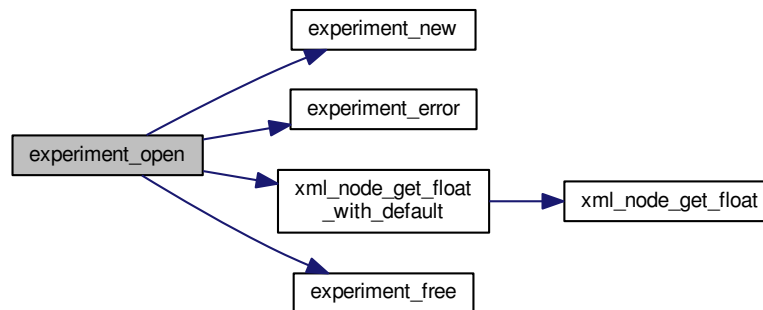
```

```

00203         experiment_error (experiment, buffer);
00204         goto exit_on_error;
00205     }
00206     else
00207         break;
00208 }
00209
00210 #if DEBUG
00211 fprintf (stderr, "experiment_open: end\n");
00212 #endif
00213 return 1;
00214
00215 exit_on_error:
00216     experiment_free (experiment);
00217 #if DEBUG
00218 fprintf (stderr, "experiment_open: end\n");
00219 #endif
00220 return 0;
00221 }

```

Here is the call graph for this function:



## 5.6 experiment.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1

```

```

00040
00045 typedef struct
00046 {
00047     char *name;
00048     char *template[MAX_NINPUTS];
00049     double weight;
00050     unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const xmlChar *template[MAX_NINPUTS];
00054
00055 // Public functions
00056 void experiment_new (Experiment * experiment);
00057 void experiment_free (Experiment * experiment);
00058 void experiment_error (Experiment * experiment, char *message);
00059 int experiment_open (Experiment * experiment, xmlNode * node,
00060                     unsigned int ninputs);
00061
00062 #endif

```

## 5.7 input.c File Reference

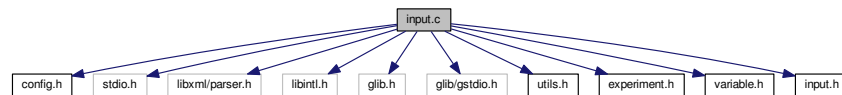
Source file to define the input functions.

```

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

```

Include dependency graph for input.c:



### Macros

- #define **\_GNU\_SOURCE**
- #define **DEBUG** 0

*Macro to debug.*

### 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** (char \*filename)  
*Function to open the input file.*

## Variables

- `Input input [1]`
- `const xmlChar * result_name = (xmlChar *) "result"`  
*Name of the result file.*
- `const xmlChar * variables_name = (xmlChar *) "variables"`  
*Name of the variables file.*

### 5.7.1 Detailed Description

Source file to define the input functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.7.2 Function Documentation

#### 5.7.2.1 void input\_error ( char \* message )

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

##### Parameters

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

Definition at line 115 of file [input.c](#).

```
00116 {
00117     char buffer[64];
00118     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00119     error_message = g_strdup (buffer);
00120 }
```

#### 5.7.2.2 int input\_open ( char \* filename )

Function to open the input file.

##### Parameters

<i>filename</i>	<a href="#">Input</a> data file name.
-----------------	---------------------------------------

##### Returns

1\_on\_success, 0\_on\_error.

Definition at line 130 of file [input.c](#).

```
00131 {
00132     char buffer2[64];
00133     xmlDoc *doc;
00134     xmlNode *node, *child;
00135     xmlChar *buffer;
```

```

00136     int error_code;
00137     unsigned int i;
00138
00139     #if DEBUG
00140         fprintf (stderr, "input_open: start\n");
00141     #endif
00142
00143     // Resetting input data
00144     buffer = NULL;
00145     input_new ();
00146
00147     // Parsing the input file
00148     #if DEBUG
00149         fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00150     #endif
00151     doc = xmlParseFile (filename);
00152     if (!doc)
00153     {
00154         input_error (gettext ("Unable to parse the input file"));
00155         goto exit_on_error;
00156     }
00157
00158     // Getting the root node
00159     #if DEBUG
00160         fprintf (stderr, "input_open: getting the root node\n");
00161     #endif
00162     node = xmlDocGetRootElement (doc);
00163     if (xmlStrcmp (node->name, XML_OPTIMIZE))
00164     {
00165         input_error (gettext ("Bad root XML node"));
00166         goto exit_on_error;
00167     }
00168
00169     // Getting result and variables file names
00170     if (!input->result)
00171     {
00172         input->result = (char *) xmlGetProp (node, XML_RESULT);
00173         if (!input->result)
00174             input->result = (char *) xmlStrdup (result_name);
00175     }
00176     if (!input->variables)
00177     {
00178         input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00179         if (!input->variables)
00180             input->variables = (char *) xmlStrdup (variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00185     if (!input->simulator)
00186     {
00187         input_error (gettext ("Bad simulator program"));
00188         goto exit_on_error;
00189     }
00190
00191     // Opening evaluator program name
00192     input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00193
00194     // Obtaining pseudo-random numbers generator seed
00195     input->seed
00196     = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
00197                                     &error_code);
00198     if (error_code)
00199     {
00200         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00201         goto exit_on_error;
00202     }
00203
00204     // Opening algorithm
00205     buffer = xmlGetProp (node, XML_ALGORITHM);
00206     if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00207     {
00208         input->algorithm = ALGORITHM_MONTE_CARLO;
00209
00210         // Obtaining simulations number
00211         input->nsimulations
00212         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00213         if (error_code)
00214         {
00215             input_error (gettext ("Bad simulations number"));
00216             goto exit_on_error;
00217         }
00218     }
00219     else if (!xmlStrcmp (buffer, XML_SWEEP))
00220         input->algorithm = ALGORITHM_SWEEP;
00221     else if (!xmlStrcmp (buffer, XML_GENETIC))

```

```

00222     {
00223         input->algorithm = ALGORITHM_GENETIC;
00224
00225         // Obtaining population
00226         if (xmlHasProp (node, XML_NPOPULATION))
00227         {
00228             input->nsimulations
00229             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00230             if (error_code || input->nsimulations < 3)
00231             {
00232                 input_error (gettext ("Invalid population number"));
00233                 goto exit_on_error;
00234             }
00235         }
00236         else
00237         {
00238             input_error (gettext ("No population number"));
00239             goto exit_on_error;
00240         }
00241
00242         // Obtaining generations
00243         if (xmlHasProp (node, XML_NGENERATIONS))
00244         {
00245             input->niterations
00246             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00247             if (error_code || !input->niterations)
00248             {
00249                 input_error (gettext ("Invalid generations number"));
00250                 goto exit_on_error;
00251             }
00252         }
00253         else
00254         {
00255             input_error (gettext ("No generations number"));
00256             goto exit_on_error;
00257         }
00258
00259         // Obtaining mutation probability
00260         if (xmlHasProp (node, XML_MUTATION))
00261         {
00262             input->mutation_ratio
00263             = xml_node_get_float (node, XML_MUTATION, &error_code);
00264             if (error_code || input->mutation_ratio < 0.
00265                 || input->mutation_ratio >= 1.)
00266             {
00267                 input_error (gettext ("Invalid mutation probability"));
00268                 goto exit_on_error;
00269             }
00270         }
00271         else
00272         {
00273             input_error (gettext ("No mutation probability"));
00274             goto exit_on_error;
00275         }
00276
00277         // Obtaining reproduction probability
00278         if (xmlHasProp (node, XML_REPRODUCTION))
00279         {
00280             input->reproduction_ratio
00281             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00282             if (error_code || input->reproduction_ratio < 0.
00283                 || input->reproduction_ratio >= 1.0)
00284             {
00285                 input_error (gettext ("Invalid reproduction probability"));
00286                 goto exit_on_error;
00287             }
00288         }
00289         else
00290         {
00291             input_error (gettext ("No reproduction probability"));
00292             goto exit_on_error;
00293         }
00294
00295         // Obtaining adaptation probability
00296         if (xmlHasProp (node, XML_ADAPTATION))
00297         {
00298             input->adaptation_ratio
00299             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00300             if (error_code || input->adaptation_ratio < 0.
00301                 || input->adaptation_ratio >= 1.)
00302             {
00303                 input_error (gettext ("Invalid adaptation probability"));
00304                 goto exit_on_error;
00305             }
00306         }
00307         else
00308         {

```

```

00309         input_error (gettext ("No adaptation probability"));
00310         goto exit_on_error;
00311     }
00312
00313     // Checking survivals
00314     i = input->mutation_ratio * input->nsimulations;
00315     i += input->reproduction_ratio * input->nsimulations;
00316     i += input->adaptation_ratio * input->nsimulations;
00317     if (i > input->nsimulations - 2)
00318     {
00319         input_error (gettext
00320             ("No enough survival entities to reproduce the population"));
00321         goto exit_on_error;
00322     }
00323
00324     else
00325     {
00326         input_error (gettext ("Unknown algorithm"));
00327         goto exit_on_error;
00328     }
00329     xmlFree (buffer);
00330     buffer = NULL;
00331
00332     if (input->algorithm == ALGORITHM_MONTE_CARLO
00333         || input->algorithm == ALGORITHM_SWEEP)
00334     {
00335
00336         // Obtaining iterations number
00337         input->niterations
00338             = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00339         if (error_code == 1)
00340             input->niterations = 1;
00341         else if (error_code)
00342         {
00343             input_error (gettext ("Bad iterations number"));
00344             goto exit_on_error;
00345         }
00346
00347         // Obtaining best number
00348         input->nbest
00349             = xml_node_get_uint_with_default (node,
00350 XML_NBEST, 1, &error_code);
00351         if (error_code || !input->nbest)
00352         {
00353             input_error (gettext ("Invalid best number"));
00354             goto exit_on_error;
00355         }
00356
00357         // Obtaining tolerance
00358         input->tolerance
00359             = xml_node_get_float_with_default (node,
00360 XML_TOLERANCE, 0.,
00361                                     &error_code);
00362         if (error_code || input->tolerance < 0.)
00363         {
00364             input_error (gettext ("Invalid tolerance"));
00365             goto exit_on_error;
00366         }
00367
00368         // Getting direction search method parameters
00369         if (xmlHasProp (node, XML_NSTEPS))
00370         {
00371             input->nsteps = xml_node_get_uint (node,
00372 XML_NSTEPS, &error_code);
00373             if (error_code || !input->nsteps)
00374             {
00375                 input_error (gettext ("Invalid steps number"));
00376                 goto exit_on_error;
00377             }
00378             buffer = xmlGetProp (node, XML_DIRECTION);
00379             if (!xmlStrcmp (buffer, XML_COORDINATES))
00380                 input->direction = DIRECTION_METHOD_COORDINATES;
00381             else if (!xmlStrcmp (buffer, XML_RANDOM))
00382             {
00383                 input->direction = DIRECTION_METHOD_RANDOM;
00384                 input->nestimates
00385                     = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00386                 if (error_code || !input->nestimates)
00387                 {
00388                     input_error (gettext ("Invalid estimates number"));
00389                     goto exit_on_error;
00390                 }
00391             }
00392             else
00393             {
00394                 input_error
00395                     (gettext ("Unknown method to estimate the direction search"));
00396             }
00397         }
00398     }

```

```

00393         goto exit_on_error;
00394     }
00395     xmlFree (buffer);
00396     buffer = NULL;
00397     input->relaxation
00398     = xml_node_get_float_with_default (node,
XML_RELAXATION,
00399                                     DEFAULT_RELAXATION, &error_code);
00400     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00401     {
00402         input_error (gettext ("Invalid relaxation parameter"));
00403         goto exit_on_error;
00404     }
00405 }
00406 else
00407     input->nsteps = 0;
00408 }
00409 // Obtaining the threshold
00410 input->threshold = xml_node_get_float_with_default (node,
XML_THRESHOLD, 0.,
00411                                                     &error_code);
00412 if (error_code)
00413 {
00414     input_error (gettext ("Invalid threshold"));
00415     goto exit_on_error;
00416 }
00417 // Reading the experimental data
00418 for (child = node->children; child; child = child->next)
00419 {
00420     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00421         break;
00422 #if DEBUG
00423     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00424 #endif
00425     input->experiment = (Experiment *)
00426         g_realloc (input->experiment,
00427                   (1 + input->nexperiments) * sizeof (Experiment));
00428     if (!input->nexperiments)
00429     {
00430         if (!experiment_open (input->experiment, child, 0))
00431             goto exit_on_error;
00432     }
00433     else
00434     {
00435         if (!experiment_open (input->experiment + input->
nexperiments, child,
00436                             input->experiment->ninputs))
00437             goto exit_on_error;
00438     }
00439     ++input->nexperiments;
00440 #if DEBUG
00441     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00442 #endif
00443 }
00444 if (!input->nexperiments)
00445 {
00446     input_error (gettext ("No optimization experiments"));
00447     goto exit_on_error;
00448 }
00449 buffer = NULL;
00450 // Reading the variables data
00451 for (; child; child = child->next)
00452 {
00453     #if DEBUG
00454         fprintf (stderr, "input_open: nvariables=%u\n", input->nvariables);
00455     #endif
00456     if (xmlStrcmp (child->name, XML_VARIABLE))
00457     {
00458         snprintf (buffer2, 64, "%s %u: %s",
00459                  gettext ("Variable"),
00460                  input->nvariables + 1, gettext ("bad XML node"));
00461         input_error (buffer2);
00462         goto exit_on_error;
00463     }
00464     input->variable = (Variable *)
00465         g_realloc (input->variable,
00466                   (1 + input->nvariables) * sizeof (Variable));
00467     if (!variable_open (input->variable + input->
nvariables, child,
00470                       input->algorithm, input->nsteps))
00471         goto exit_on_error;
00472     ++input->nvariables;
00473 }
00474 if (!input->nvariables)

```

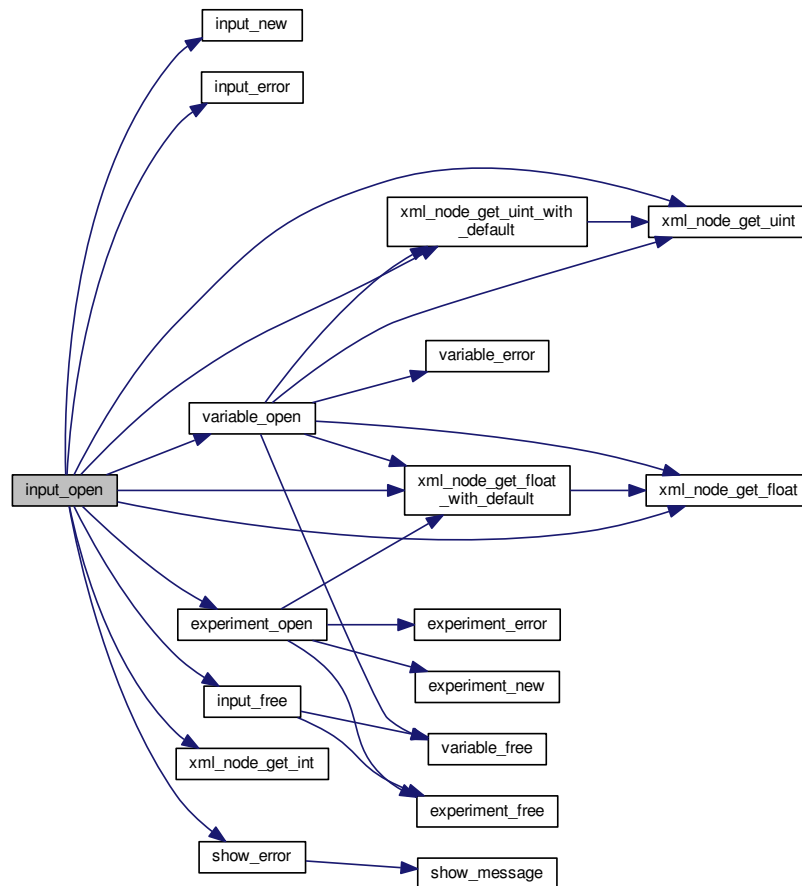


```

00475     {
00476         input_error (gettext ("No optimization variables"));
00477         goto exit_on_error;
00478     }
00479     buffer = NULL;
00480
00481     // Obtaining the error norm
00482     if (xmlHasProp (node, XML_NORM))
00483     {
00484         buffer = xmlGetProp (node, XML_NORM);
00485         if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00486             input->norm = ERROR_NORM_EUCLIDIAN;
00487         else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00488             input->norm = ERROR_NORM_MAXIMUM;
00489         else if (!xmlStrcmp (buffer, XML_P))
00490         {
00491             input->norm = ERROR_NORM_P;
00492             input->p = xml_node_get_float (node, XML_P, &error_code);
00493             if (!error_code)
00494             {
00495                 input_error (gettext ("Bad P parameter"));
00496                 goto exit_on_error;
00497             }
00498         }
00499         else if (!xmlStrcmp (buffer, XML_TAXICAB))
00500             input->norm = ERROR_NORM_TAXICAB;
00501         else
00502         {
00503             input_error (gettext ("Unknown error norm"));
00504             goto exit_on_error;
00505         }
00506         xmlFree (buffer);
00507     }
00508     else
00509         input->norm = ERROR_NORM_EUCLIDIAN;
00510
00511     // Getting the working directory
00512     input->directory = g_path_get_dirname (filename);
00513     input->name = g_path_get_basename (filename);
00514
00515     // Closing the XML document
00516     xmlFreeDoc (doc);
00517
00518     #if DEBUG
00519     fprintf (stderr, "input_open: end\n");
00520     #endif
00521     return 1;
00522
00523 exit_on_error:
00524     xmlFree (buffer);
00525     xmlFreeDoc (doc);
00526     show_error (error_message);
00527     g_free (error_message);
00528     input_free ();
00529     #if DEBUG
00530     fprintf (stderr, "input_open: end\n");
00531     #endif
00532     return 0;
00533 }

```

Here is the call graph for this function:



## 5.8 input.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
  
```

```

00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <glib/gstdio.h>
00045 #include "utils.h"
00046 #include "experiment.h"
00047 #include "variable.h"
00048 #include "input.h"
00049
00050 #define DEBUG 0
00051
00052 Input input[1];
00053
00054 const xmlChar *result_name = (xmlChar *) "result";
00055 const xmlChar *variables_name = (xmlChar *) "variables";
00056
00057 void
00058 input_new ()
00059 {
00060     #if DEBUG
00061         fprintf (stderr, "input_new: start\n");
00062     #endif
00063     input->nvariables = input->nexperiments = input->nsteps = 0;
00064     input->simulator = input->evaluator = input->directory = input->
        name
00065     = input->result = input->variables = NULL;
00066     input->experiment = NULL;
00067     input->variable = NULL;
00068     #if DEBUG
00069         fprintf (stderr, "input_new: end\n");
00070     #endif
00071 }
00072
00073 void
00074 input_free ()
00075 {
00076     unsigned int i;
00077     #if DEBUG
00078         fprintf (stderr, "input_free: start\n");
00079     #endif
00080     g_free (input->name);
00081     g_free (input->directory);
00082     for (i = 0; i < input->nexperiments; ++i)
00083         experiment_free (input->experiment + i);
00084     g_free (input->experiment);
00085     for (i = 0; i < input->nvariables; ++i)
00086         variable_free (input->variable + i);
00087     g_free (input->variable);
00088     xmlFree (input->evaluator);
00089     xmlFree (input->simulator);
00090     xmlFree (input->result);
00091     xmlFree (input->variables);
00092     input->nexperiments = input->nvariables = input->nsteps = 0;
00093     #if DEBUG
00094         fprintf (stderr, "input_free: end\n");
00095     #endif
00096 }
00097
00098 void
00099 input_error (char *message)
00100 {
00101     char buffer[64];
00102     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00103     error_message = g_strdup (buffer);
00104 }
00105
00106 int
00107 input_open (char *filename)
00108 {
00109     char buffer2[64];
00110     xmlDoc *doc;
00111     xmlNode *node, *child;
00112     xmlChar *buffer;
00113     int error_code;
00114     unsigned int i;
00115     #if DEBUG
00116         fprintf (stderr, "input_open: start\n");
00117     #endif
00118     // Resetting input data
00119     buffer = NULL;

```

```

00145     input_new ();
00146
00147     // Parsing the input file
00148     #if DEBUG
00149     fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00150     #endif
00151     doc = xmlParseFile (filename);
00152     if (!doc)
00153     {
00154         input_error (gettext ("Unable to parse the input file"));
00155         goto exit_on_error;
00156     }
00157
00158     // Getting the root node
00159     #if DEBUG
00160     fprintf (stderr, "input_open: getting the root node\n");
00161     #endif
00162     node = xmlDocGetRootElement (doc);
00163     if (xmlStrcmp (node->name, XML_OPTIMIZE))
00164     {
00165         input_error (gettext ("Bad root XML node"));
00166         goto exit_on_error;
00167     }
00168
00169     // Getting result and variables file names
00170     if (!input->result)
00171     {
00172         input->result = (char *) xmlGetProp (node, XML_RESULT);
00173         if (!input->result)
00174             input->result = (char *) xmlStrdup (result_name);
00175     }
00176     if (!input->variables)
00177     {
00178         input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00179         if (!input->variables)
00180             input->variables = (char *) xmlStrdup (variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00185     if (!input->simulator)
00186     {
00187         input_error (gettext ("Bad simulator program"));
00188         goto exit_on_error;
00189     }
00190
00191     // Opening evaluator program name
00192     input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00193
00194     // Obtaining pseudo-random numbers generator seed
00195     input->seed
00196     = xml_node_get_uint_with_default (node,
00197     XML_SEED, DEFAULT_RANDOM_SEED,
00198     &error_code);
00199     if (error_code)
00200     {
00201         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }
00204
00205     // Opening algorithm
00206     buffer = xmlGetProp (node, XML_ALGORITHM);
00207     if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00208     {
00209         input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211         // Obtaining simulations number
00212         input->nsimulations
00213         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00214         if (error_code)
00215         {
00216             input_error (gettext ("Bad simulations number"));
00217             goto exit_on_error;
00218         }
00219     }
00220     else if (!xmlStrcmp (buffer, XML_SWEEP))
00221         input->algorithm = ALGORITHM_SWEEP;
00222     else if (!xmlStrcmp (buffer, XML_GENETIC))
00223     {
00224         input->algorithm = ALGORITHM_GENETIC;
00225
00226         // Obtaining population
00227         if (xmlHasProp (node, XML_NPOPULATION))
00228         {
00229             input->nsimulations
00230             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00231             if (error_code || input->nsimulations < 3)

```

```

00231         {
00232             input_error (gettext ("Invalid population number"));
00233             goto exit_on_error;
00234         }
00235     }
00236     else
00237     {
00238         input_error (gettext ("No population number"));
00239         goto exit_on_error;
00240     }
00241
00242     // Obtaining generations
00243     if (xmlHasProp (node, XML_NGENERATIONS))
00244     {
00245         input->niterations
00246         = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00247         if (error_code || !input->niterations)
00248         {
00249             input_error (gettext ("Invalid generations number"));
00250             goto exit_on_error;
00251         }
00252     }
00253     else
00254     {
00255         input_error (gettext ("No generations number"));
00256         goto exit_on_error;
00257     }
00258
00259     // Obtaining mutation probability
00260     if (xmlHasProp (node, XML_MUTATION))
00261     {
00262         input->mutation_ratio
00263         = xml_node_get_float (node, XML_MUTATION, &error_code);
00264         if (error_code || input->mutation_ratio < 0.
00265             || input->mutation_ratio >= 1.)
00266         {
00267             input_error (gettext ("Invalid mutation probability"));
00268             goto exit_on_error;
00269         }
00270     }
00271     else
00272     {
00273         input_error (gettext ("No mutation probability"));
00274         goto exit_on_error;
00275     }
00276
00277     // Obtaining reproduction probability
00278     if (xmlHasProp (node, XML_REPRODUCTION))
00279     {
00280         input->reproduction_ratio
00281         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00282         if (error_code || input->reproduction_ratio < 0.
00283             || input->reproduction_ratio >= 1.0)
00284         {
00285             input_error (gettext ("Invalid reproduction probability"));
00286             goto exit_on_error;
00287         }
00288     }
00289     else
00290     {
00291         input_error (gettext ("No reproduction probability"));
00292         goto exit_on_error;
00293     }
00294
00295     // Obtaining adaptation probability
00296     if (xmlHasProp (node, XML_ADAPTATION))
00297     {
00298         input->adaptation_ratio
00299         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00300         if (error_code || input->adaptation_ratio < 0.
00301             || input->adaptation_ratio >= 1.)
00302         {
00303             input_error (gettext ("Invalid adaptation probability"));
00304             goto exit_on_error;
00305         }
00306     }
00307     else
00308     {
00309         input_error (gettext ("No adaptation probability"));
00310         goto exit_on_error;
00311     }
00312
00313     // Checking survivals
00314     i = input->mutation_ratio * input->nsimulations;
00315     i += input->reproduction_ratio * input->nsimulations;
00316     i += input->adaptation_ratio * input->nsimulations;
00317     if (i > input->nsimulations - 2)

```

```

00318     {
00319         input_error (gettext
00320             ("No enough survival entities to reproduce the population"));
00321         goto exit_on_error;
00322     }
00323 }
00324 else
00325 {
00326     input_error (gettext ("Unknown algorithm"));
00327     goto exit_on_error;
00328 }
00329 xmlFree (buffer);
00330 buffer = NULL;
00331
00332 if (input->algorithm == ALGORITHM_MONTE_CARLO
00333     || input->algorithm == ALGORITHM_SWEEP)
00334 {
00335     // Obtaining iterations number
00336     input->niterations
00337         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00338     if (error_code == 1)
00339         input->niterations = 1;
00340     else if (error_code)
00341     {
00342         input_error (gettext ("Bad iterations number"));
00343         goto exit_on_error;
00344     }
00345
00346     // Obtaining best number
00347     input->nbest
00348         = xml_node_get_uint_with_default (node,
00349     XML_NBEST, 1, &error_code);
00350     if (error_code || !input->nbest)
00351     {
00352         input_error (gettext ("Invalid best number"));
00353         goto exit_on_error;
00354     }
00355
00356     // Obtaining tolerance
00357     input->tolerance
00358         = xml_node_get_float_with_default (node,
00359     XML_TOLERANCE, 0.,
00360                                     &error_code);
00361     if (error_code || input->tolerance < 0.)
00362     {
00363         input_error (gettext ("Invalid tolerance"));
00364         goto exit_on_error;
00365     }
00366
00367     // Getting direction search method parameters
00368     if (xmlHasProp (node, XML_NSTEPS))
00369     {
00370         input->nsteps = xml_node_get_uint (node,
00371     XML_NSTEPS, &error_code);
00372         if (error_code || !input->nsteps)
00373         {
00374             input_error (gettext ("Invalid steps number"));
00375             goto exit_on_error;
00376         }
00377         buffer = xmlGetProp (node, XML_DIRECTION);
00378         if (!xmlStrcmp (buffer, XML_COORDINATES))
00379             input->direction = DIRECTION_METHOD_COORDINATES;
00380         else if (!xmlStrcmp (buffer, XML_RANDOM))
00381             input->direction = DIRECTION_METHOD_RANDOM;
00382         input->nestimates
00383             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00384         if (error_code || !input->nestimates)
00385         {
00386             input_error (gettext ("Invalid estimates number"));
00387             goto exit_on_error;
00388         }
00389     }
00390     else
00391     {
00392         input_error
00393             (gettext ("Unknown method to estimate the direction search"));
00394         goto exit_on_error;
00395     }
00396     xmlFree (buffer);
00397     buffer = NULL;
00398     input->relaxation
00399         = xml_node_get_float_with_default (node,
00400     XML_RELAXATION,
00401                                     DEFAULT_RELAXATION, &error_code);
00402     if (error_code || input->relaxation < 0. || input->

```

```

        relaxation > 2.)
00401     {
00402         input_error (gettext ("Invalid relaxation parameter"));
00403         goto exit_on_error;
00404     }
00405 }
00406 else
00407     input->nsteps = 0;
00408 }
00409 // Obtaining the threshold
00410 input->threshold = xml_node_get_float_with_default (node,
XML_THRESHOLD, 0.,
00411                                                     &error_code);
00412 if (error_code)
00413 {
00414     input_error (gettext ("Invalid threshold"));
00415     goto exit_on_error;
00416 }
00417
00418 // Reading the experimental data
00419 for (child = node->children; child; child = child->next)
00420 {
00421     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00422         break;
00423 #if DEBUG
00424     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00425 #endif
00426     input->experiment = (Experiment *)
00427         g_realloc (input->experiment,
00428                   (1 + input->nexperiments) * sizeof (Experiment));
00429     if (!input->nexperiments)
00430     {
00431         if (!experiment_open (input->experiment, child, 0))
00432             goto exit_on_error;
00433     }
00434     else
00435     {
00436         if (!experiment_open (input->experiment + input->
nexperiments, child,
00437                               input->experiment->ninputs))
00438             goto exit_on_error;
00439     }
00440     ++input->nexperiments;
00441 #if DEBUG
00442     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00443 #endif
00444 }
00445 if (!input->nexperiments)
00446 {
00447     input_error (gettext ("No optimization experiments"));
00448     goto exit_on_error;
00449 }
00450 buffer = NULL;
00451
00452 // Reading the variables data
00453 for (; child; child = child->next)
00454 {
00455     #if DEBUG
00456         fprintf (stderr, "input_open: nvariables=%u\n", input->nvariables);
00457     #endif
00458     if (xmlStrcmp (child->name, XML_VARIABLE))
00459     {
00460         snprintf (buffer2, 64, "%s %u: %s",
00461                  gettext ("Variable"),
00462                  input->nvariables + 1, gettext ("bad XML node"));
00463         input_error (buffer2);
00464         goto exit_on_error;
00465     }
00466     input->variable = (Variable *)
00467         g_realloc (input->variable,
00468                   (1 + input->nvariables) * sizeof (Variable));
00469     if (!variable_open (input->variable + input->
nvariables, child,
00470                         input->algorithm, input->nsteps))
00471         goto exit_on_error;
00472     ++input->nvariables;
00473 }
00474 if (!input->nvariables)
00475 {
00476     input_error (gettext ("No optimization variables"));
00477     goto exit_on_error;
00478 }
00479 buffer = NULL;
00480
00481 // Obtaining the error norm
00482 if (xmlHasProp (node, XML_NORM))
00483 {

```

```

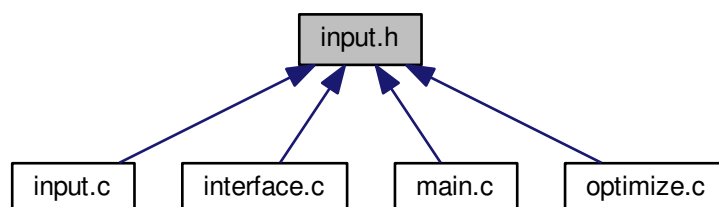
00484     buffer = xmlGetProp (node, XML_NORM);
00485     if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00486         input->norm = ERROR_NORM_EUCLIDIAN;
00487     else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00488         input->norm = ERROR_NORM_MAXIMUM;
00489     else if (!xmlStrcmp (buffer, XML_P))
00490     {
00491         input->norm = ERROR_NORM_P;
00492         input->p = xml_node_get_float (node, XML_P, &error_code);
00493         if (!error_code)
00494         {
00495             input_error (gettext ("Bad P parameter"));
00496             goto exit_on_error;
00497         }
00498     }
00499     else if (!xmlStrcmp (buffer, XML_TAXICAB))
00500         input->norm = ERROR_NORM_TAXICAB;
00501     else
00502     {
00503         input_error (gettext ("Unknown error norm"));
00504         goto exit_on_error;
00505     }
00506     xmlFree (buffer);
00507 }
00508 else
00509     input->norm = ERROR_NORM_EUCLIDIAN;
00510
00511 // Getting the working directory
00512 input->directory = g_path_get_dirname (filename);
00513 input->name = g_path_get_basename (filename);
00514
00515 // Closing the XML document
00516 xmlFreeDoc (doc);
00517
00518 #if DEBUG
00519 fprintf (stderr, "input_open: end\n");
00520 #endif
00521 return 1;
00522
00523 exit_on_error:
00524     xmlFree (buffer);
00525     xmlFreeDoc (doc);
00526     show_error (error_message);
00527     g_free (error_message);
00528     input_free ();
00529 #if DEBUG
00530 fprintf (stderr, "input_open: end\n");
00531 #endif
00532 return 0;
00533 }

```

## 5.9 input.h File Reference

Header file to define the input functions.

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





## Data Structures

- struct [Input](#)

*Struct to define the optimization input file.*

## Enumerations

- enum [DirectionMethod](#) { [DIRECTION\\_METHOD\\_COORDINATES](#) = 0, [DIRECTION\\_METHOD\\_RANDOM](#) = 1 }

*Enum to define the methods to estimate the direction search.*

- enum [ErrorNorm](#) { [ERROR\\_NORM\\_EUCLIDIAN](#) = 0, [ERROR\\_NORM\\_MAXIMUM](#) = 1, [ERROR\\_NORM\\_P](#) = 2, [ERROR\\_NORM\\_TAXICAB](#) = 3 }

*Enum to define the error norm.*

## Functions

- void [input\\_new](#) ()

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

- void [input\\_free](#) ()

*Function to free the memory of the input file data.*

- void [input\\_error](#) (char \*message)

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

- int [input\\_open](#) (char \*filename)

*Function to open the input file.*

## Variables

- [Input](#) [input](#) [1]

- const xmlChar \* [result\\_name](#)

*Name of the result file.*

- const xmlChar \* [variables\\_name](#)

*Name of the variables file.*

### 5.9.1 Detailed Description

Header file to define the input functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

## 5.9.2 Enumeration Type Documentation

### 5.9.2.1 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 };
```

### 5.9.2.2 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 };
```

## 5.9.3 Function Documentation

### 5.9.3.1 void input\_error ( char \* message )

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

Parameters

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

Definition at line 115 of file [input.c](#).

```
00116 {
00117     char buffer[64];
00118     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00119     error_message = g_strdup (buffer);
00120 }
```

### 5.9.3.2 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 130 of file [input.c](#).

```

00131 {
00132     char buffer2[64];
00133     xmlDoc *doc;
00134     xmlNode *node, *child;
00135     xmlChar *buffer;
00136     int error_code;
00137     unsigned int i;
00138
00139     #if DEBUG
00140         fprintf (stderr, "input_open: start\n");
00141     #endif
00142
00143     // Resetting input data
00144     buffer = NULL;
00145     input_new ();
00146
00147     // Parsing the input file
00148     #if DEBUG
00149         fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00150     #endif
00151     doc = xmlParseFile (filename);
00152     if (!doc)
00153     {
00154         input_error (gettext ("Unable to parse the input file"));
00155         goto exit_on_error;
00156     }
00157
00158     // Getting the root node
00159     #if DEBUG
00160         fprintf (stderr, "input_open: getting the root node\n");
00161     #endif
00162     node = xmlDocGetRootElement (doc);
00163     if (xmlStrcmp (node->name, XML_OPTIMIZE))
00164     {
00165         input_error (gettext ("Bad root XML node"));
00166         goto exit_on_error;
00167     }
00168
00169     // Getting result and variables file names
00170     if (!input->result)
00171     {
00172         input->result = (char *) xmlGetProp (node, XML_RESULT);
00173         if (!input->result)
00174             input->result = (char *) xmlStrdup (result_name);
00175     }
00176     if (!input->variables)
00177     {
00178         input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00179         if (!input->variables)
00180             input->variables = (char *) xmlStrdup (variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00185     if (!input->simulator)
00186     {
00187         input_error (gettext ("Bad simulator program"));
00188         goto exit_on_error;
00189     }
00190
00191     // Opening evaluator program name
00192     input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00193
00194     // Obtaining pseudo-random numbers generator seed
00195     input->seed
00196         = xml_node_get_uint_with_default (node,
00197         XML_SEED, DEFAULT_RANDOM_SEED,
00198         &error_code);
00199     if (error_code)
00200     {
00201         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }

```

```

00203
00204 // Opening algorithm
00205 buffer = xmlGetProp (node, XML_ALGORITHM);
00206 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00207 {
00208     input->algorithm = ALGORITHM_MONTE_CARLO;
00209
00210     // Obtaining simulations number
00211     input->nsimulations
00212     = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00213     if (error_code)
00214     {
00215         input_error (gettext ("Bad simulations number"));
00216         goto exit_on_error;
00217     }
00218 }
00219 else if (!xmlStrcmp (buffer, XML_SWEEP))
00220     input->algorithm = ALGORITHM_SWEEP;
00221 else if (!xmlStrcmp (buffer, XML_GENETIC))
00222 {
00223     input->algorithm = ALGORITHM_GENETIC;
00224
00225     // Obtaining population
00226     if (xmlHasProp (node, XML_NPOPULATION))
00227     {
00228         input->nsimulations
00229         = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00230         if (error_code || input->nsimulations < 3)
00231         {
00232             input_error (gettext ("Invalid population number"));
00233             goto exit_on_error;
00234         }
00235     }
00236     else
00237     {
00238         input_error (gettext ("No population number"));
00239         goto exit_on_error;
00240     }
00241
00242     // Obtaining generations
00243     if (xmlHasProp (node, XML_NGENERATIONS))
00244     {
00245         input->niterations
00246         = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00247         if (error_code || !input->niterations)
00248         {
00249             input_error (gettext ("Invalid generations number"));
00250             goto exit_on_error;
00251         }
00252     }
00253     else
00254     {
00255         input_error (gettext ("No generations number"));
00256         goto exit_on_error;
00257     }
00258
00259     // Obtaining mutation probability
00260     if (xmlHasProp (node, XML_MUTATION))
00261     {
00262         input->mutation_ratio
00263         = xml_node_get_float (node, XML_MUTATION, &error_code);
00264         if (error_code || input->mutation_ratio < 0.
00265             || input->mutation_ratio >= 1.)
00266         {
00267             input_error (gettext ("Invalid mutation probability"));
00268             goto exit_on_error;
00269         }
00270     }
00271     else
00272     {
00273         input_error (gettext ("No mutation probability"));
00274         goto exit_on_error;
00275     }
00276
00277     // Obtaining reproduction probability
00278     if (xmlHasProp (node, XML_REPRODUCTION))
00279     {
00280         input->reproduction_ratio
00281         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00282         if (error_code || input->reproduction_ratio < 0.
00283             || input->reproduction_ratio >= 1.0)
00284         {
00285             input_error (gettext ("Invalid reproduction probability"));
00286             goto exit_on_error;
00287         }
00288     }
00289     else

```

```

00290     {
00291         input_error (gettext ("No reproduction probability"));
00292         goto exit_on_error;
00293     }
00294
00295     // Obtaining adaptation probability
00296     if (xmlHasProp (node, XML_ADAPTATION))
00297     {
00298         input->adaptation_ratio
00299         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00300         if (error_code || input->adaptation_ratio < 0.
00301             || input->adaptation_ratio >= 1.)
00302         {
00303             input_error (gettext ("Invalid adaptation probability"));
00304             goto exit_on_error;
00305         }
00306     }
00307     else
00308     {
00309         input_error (gettext ("No adaptation probability"));
00310         goto exit_on_error;
00311     }
00312
00313     // Checking survivals
00314     i = input->mutation_ratio * input->nsimulations;
00315     i += input->reproduction_ratio * input->nsimulations;
00316     i += input->adaptation_ratio * input->nsimulations;
00317     if (i > input->nsimulations - 2)
00318     {
00319         input_error (gettext
00320             ("No enough survival entities to reproduce the population"));
00321         goto exit_on_error;
00322     }
00323 }
00324 else
00325 {
00326     input_error (gettext ("Unknown algorithm"));
00327     goto exit_on_error;
00328 }
00329 xmlFree (buffer);
00330 buffer = NULL;
00331
00332 if (input->algorithm == ALGORITHM_MONTE_CARLO
00333     || input->algorithm == ALGORITHM_SWEEP)
00334 {
00335     // Obtaining iterations number
00336     input->niterations
00337     = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00338     if (error_code == 1)
00339         input->niterations = 1;
00340     else if (error_code)
00341     {
00342         input_error (gettext ("Bad iterations number"));
00343         goto exit_on_error;
00344     }
00345 }
00346
00347 // Obtaining best number
00348 input->nbest
00349 = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
00350 if (error_code || !input->nbest)
00351 {
00352     input_error (gettext ("Invalid best number"));
00353     goto exit_on_error;
00354 }
00355
00356 // Obtaining tolerance
00357 input->tolerance
00358 = xml_node_get_float_with_default (node,
XML_TOLERANCE, 0.,
                                &error_code);
00359 if (error_code || input->tolerance < 0.)
00360 {
00361     input_error (gettext ("Invalid tolerance"));
00362     goto exit_on_error;
00363 }
00364
00365 // Getting direction search method parameters
00366 if (xmlHasProp (node, XML_NSTEPS))
00367 {
00368     input->nsteps = xml_node_get_uint (node,
XML_NSTEPS, &error_code);
00369     if (error_code || !input->nsteps)
00370     {
00371         input_error (gettext ("Invalid steps number"));
00372         goto exit_on_error;
00373     }

```

```

00374     }
00375     buffer = xmlGetProp (node, XML_DIRECTION);
00376     if (!xmlStrcmp (buffer, XML_COORDINATES))
00377         input->direction = DIRECTION_METHOD_COORDINATES;
00378     else if (!xmlStrcmp (buffer, XML_RANDOM))
00379     {
00380         input->direction = DIRECTION_METHOD_RANDOM;
00381         input->nestimates
00382             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00383         if (error_code || !input->nestimates)
00384         {
00385             input_error (gettext ("Invalid estimates number"));
00386             goto exit_on_error;
00387         }
00388     }
00389     else
00390     {
00391         input_error
00392             (gettext ("Unknown method to estimate the direction search"));
00393         goto exit_on_error;
00394     }
00395     xmlFree (buffer);
00396     buffer = NULL;
00397     input->relaxation
00398         = xml_node_get_float_with_default (node,
00399 XML_RELAXATION,
00400                                         DEFAULT_RELAXATION, &error_code);
00401     if (error_code || input->relaxation < 0. || input->
00402 relaxation > 2.)
00403     {
00404         input_error (gettext ("Invalid relaxation parameter"));
00405         goto exit_on_error;
00406     }
00407     else
00408         input->nsteps = 0;
00409     // Obtaining the threshold
00410     input->threshold = xml_node_get_float_with_default (node,
00411 XML_THRESHOLD, 0.,
00412                                                         &error_code);
00413     if (error_code)
00414     {
00415         input_error (gettext ("Invalid threshold"));
00416         goto exit_on_error;
00417     }
00418     // Reading the experimental data
00419     for (child = node->children; child; child = child->next)
00420     {
00421         if (xmlStrcmp (child->name, XML_EXPERIMENT))
00422             break;
00423 #if DEBUG
00424         fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00425 #endif
00426         input->experiment = (Experiment *)
00427             g_realloc (input->experiment,
00428                       (1 + input->nexperiments) * sizeof (Experiment));
00429         if (!input->nexperiments)
00430         {
00431             if (!experiment_open (input->experiment, child, 0))
00432                 goto exit_on_error;
00433         }
00434         else
00435         {
00436             if (!experiment_open (input->experiment + input->
00437 nexperiments, child,
00438                                 input->experiment->ninputs))
00439                 goto exit_on_error;
00440             ++input->nexperiments;
00441 #if DEBUG
00442             fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00443 #endif
00444         }
00445         if (!input->nexperiments)
00446         {
00447             input_error (gettext ("No optimization experiments"));
00448             goto exit_on_error;
00449         }
00450         buffer = NULL;
00451     }
00452     // Reading the variables data
00453     for (; child; child = child->next)
00454     {
00455 #if DEBUG
00456         fprintf (stderr, "input_open: nvariables=%u\n", input->nvariables);

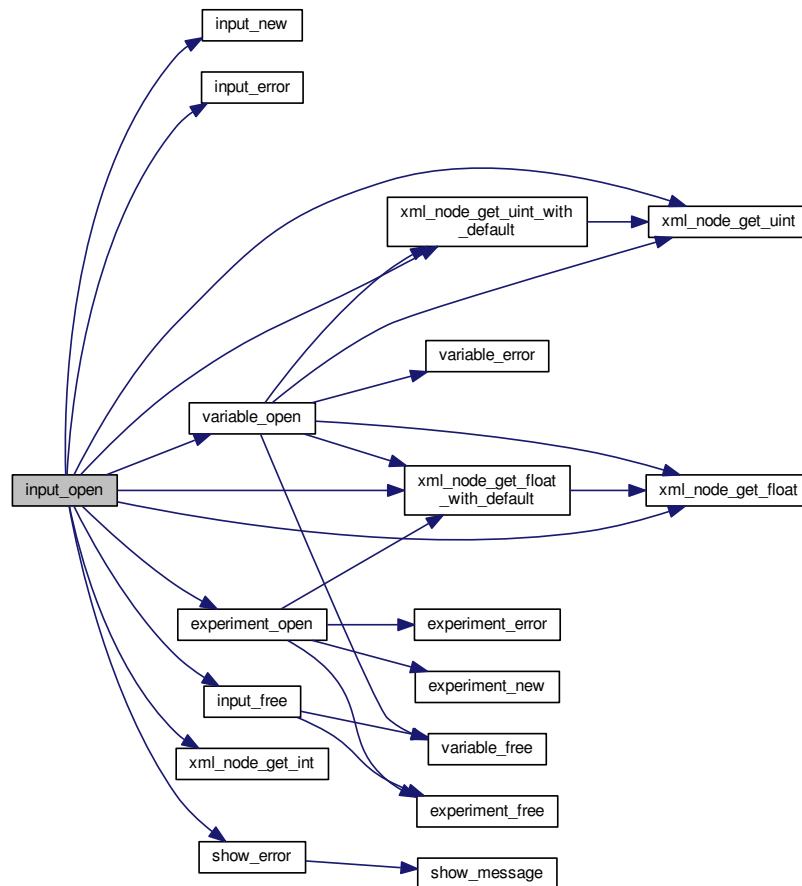
```

```

00457 #endif
00458     if (xmlStrcmp (child->name, XML_VARIABLE))
00459     {
00460         snprintf (buffer2, 64, "%s %u: %s",
00461             gettext ("Variable"),
00462             input->nvariables + 1, gettext ("bad XML node"));
00463         input_error (buffer2);
00464         goto exit_on_error;
00465     }
00466     input->variable = (Variable *)
00467         g_realloc (input->variable,
00468             (1 + input->nvariables) * sizeof (Variable));
00469     if (!variable_open (input->variable + input->
nvariables, child,
00470         input->algorithm, input->nsteps))
00471         goto exit_on_error;
00472     ++input->nvariables;
00473 }
00474 if (!input->nvariables)
00475 {
00476     input_error (gettext ("No optimization variables"));
00477     goto exit_on_error;
00478 }
00479 buffer = NULL;
00480
00481 // Obtaining the error norm
00482 if (xmlHasProp (node, XML_NORM))
00483 {
00484     buffer = xmlGetProp (node, XML_NORM);
00485     if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
00486         input->norm = ERROR_NORM_EUCLIDIAN;
00487     else if (!xmlStrcmp (buffer, XML_MAXIMUM))
00488         input->norm = ERROR_NORM_MAXIMUM;
00489     else if (!xmlStrcmp (buffer, XML_P))
00490     {
00491         input->norm = ERROR_NORM_P;
00492         input->p = xml_node_get_float (node, XML_P, &error_code);
00493         if (!error_code)
00494         {
00495             input_error (gettext ("Bad P parameter"));
00496             goto exit_on_error;
00497         }
00498     }
00499     else if (!xmlStrcmp (buffer, XML_TAXICAB))
00500         input->norm = ERROR_NORM_TAXICAB;
00501     else
00502     {
00503         input_error (gettext ("Unknown error norm"));
00504         goto exit_on_error;
00505     }
00506     xmlFree (buffer);
00507 }
00508 else
00509     input->norm = ERROR_NORM_EUCLIDIAN;
00510
00511 // Getting the working directory
00512 input->directory = g_path_get_dirname (filename);
00513 input->name = g_path_get_basename (filename);
00514
00515 // Closing the XML document
00516 xmlFreeDoc (doc);
00517
00518 #if DEBUG
00519 fprintf (stderr, "input_open: end\n");
00520 #endif
00521 return 1;
00522
00523 exit_on_error:
00524 xmlFree (buffer);
00525 xmlFreeDoc (doc);
00526 show_error (error_message);
00527 g_free (error_message);
00528 input_free ();
00529 #if DEBUG
00530 fprintf (stderr, "input_open: end\n");
00531 #endif
00532 return 0;
00533 }

```

Here is the call graph for this function:



## 5.10 input.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
  
```



```

00030 */
00031
00038 #ifndef INPUT__H
00039 #define INPUT__H 1
00040
00045 enum DirectionMethod
00046 {
00047     DIRECTION_METHOD_COORDINATES = 0,
00048     DIRECTION_METHOD_RANDOM = 1,
00049 };
00050
00055 enum ErrorNorm
00056 {
00057     ERROR_NORM_EUCLIDIAN = 0,
00059     ERROR_NORM_MAXIMUM = 1,
00061     ERROR_NORM_P = 2,
00063     ERROR_NORM_TAXICAB = 3
00065 };
00066
00071 typedef struct
00072 {
00073     Experiment *experiment;
00074     Variable *variable;
00075     char *result;
00076     char *variables;
00077     char *simulator;
00078     char *evaluator;
00080     char *directory;
00081     char *name;
00082     double tolerance;
00083     double mutation_ratio;
00084     double reproduction_ratio;
00085     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 } Input;
00104
00105 extern Input input[1];
00106 extern const xmlChar *result_name;
00107 extern const xmlChar *variables_name;
00108
00109 // Public functions
00110 void input_new ();
00111 void input_free ();
00112 void input_error (char *message);
00113 int input_open (char *filename);
00114
00115 #endif
00116

```

## 5.11 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 <alloca.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 GNU_SOURCE`
- `#define DEBUG 0`  
*Macro to debug.*
- `#define INPUT_FILE "test-ga.xml"`  
*Macro to define the initial input file.*

## Functions

- void `input_save_direction` (xmlNode \*node)  
*Function to save the direction search method data in a XML node.*
- 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](#) ()

*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.*

### 5.11.1 Detailed Description

Source file to define the graphical interface functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.11.2 Function Documentation

#### 5.11.2.1 void input\_save ( char \* filename )

Function to save the input file.

#### Parameters

<i>filename</i>	<a href="#">Input</a> file name.
-----------------	----------------------------------

Definition at line 204 of file [interface.c](#).

```

00205 {
00206     unsigned int i, j;
00207     char *buffer;
00208     xmlDoc *doc;
00209     xmlNode *node, *child;
00210     GFile *file, *file2;
00211
00212     #if DEBUG
00213         fprintf (stderr, "input_save: start\n");
00214     #endif
00215
00216     // Getting the input file directory
00217     input->name = g_path_get_basename (filename);
00218     input->directory = g_path_get_dirname (filename);
00219     file = g_file_new_for_path (input->directory);
00220
00221     // Opening the input file

```

```

00222 doc = xmlNewDoc ((const xmlChar *) "1.0");
00223
00224 // Setting root XML node
00225 node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00226 xmlDocSetRootElement (doc, node);
00227
00228 // Adding properties to the root XML node
00229 if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00230     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00231 if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
00232     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
00233 file2 = g_file_new_for_path (input->simulator);
00234 buffer = g_file_get_relative_path (file, file2);
00235 g_object_unref (file2);
00236 xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00237 g_free (buffer);
00238 if (input->evaluator)
00239 {
00240     file2 = g_file_new_for_path (input->evaluator);
00241     buffer = g_file_get_relative_path (file, file2);
00242     g_object_unref (file2);
00243     if (xmlStrlen ((xmlChar *) buffer))
00244         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00245     g_free (buffer);
00246 }
00247 if (input->seed != DEFAULT_RANDOM_SEED)
00248     xml_node_set_uint (node, XML_SEED, input->seed);
00249
00250 // Setting the algorithm
00251 buffer = (char *) g_malloc (64);
00252 switch (input->algorithm)
00253 {
00254     case ALGORITHM_MONTE_CARLO:
00255         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00256         snprintf (buffer, 64, "%u", input->nsimulations);
00257         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00258         snprintf (buffer, 64, "%u", input->niterations);
00259         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00260         snprintf (buffer, 64, "%.3lg", input->tolerance);
00261         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00262         snprintf (buffer, 64, "%u", input->nbest);
00263         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00264         input_save_direction (node);
00265         break;
00266     case ALGORITHM_SWEEP:
00267         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00268         snprintf (buffer, 64, "%u", input->niterations);
00269         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00270         snprintf (buffer, 64, "%.3lg", input->tolerance);
00271         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00272         snprintf (buffer, 64, "%u", input->nbest);
00273         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00274         input_save_direction (node);
00275         break;
00276     default:
00277         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00278         snprintf (buffer, 64, "%u", input->nsimulations);
00279         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00280         snprintf (buffer, 64, "%u", input->niterations);
00281         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00282         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00283         xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00284         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00285         xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00286         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00287         xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00288         break;
00289 }
00290 g_free (buffer);
00291 if (input->threshold != 0.)
00292     xml_node_set_float (node, XML_THRESHOLD, input->
threshold);
00293
00294 // Setting the experimental data
00295 for (i = 0; i < input->nexperiments; ++i)
00296 {
00297     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00298     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i].
name);
00299     if (input->experiment[i].weight != 1.)
00300         xml_node_set_float (child, XML_WEIGHT, input->
experiment[i].weight);
00301     for (j = 0; j < input->experiment->ninputs; ++j)
00302         xmlSetProp (child, template[j],
(xmlChar *) input->experiment[i].template[j]);
00303 }
00304
00305

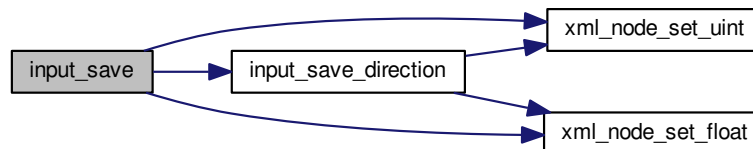
```

```

00306 // Setting the variables data
00307 for (i = 0; i < input->nvariables; ++i)
00308 {
00309     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00310     xmlSetProp (child, XML_NAME, (xmlChar *) input->variable[i].
name);
00311     xml_node_set_float (child, XML_MINIMUM, input->
variable[i].rangemin);
00312     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00313         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->variable[i].rangeminabs);
00314     xml_node_set_float (child, XML_MAXIMUM, input->
variable[i].rangemax);
00315     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00316         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->variable[i].rangemaxabs);
00317     if (input->variable[i].precision != DEFAULT_PRECISION)
00318         xml_node_set_uint (child, XML_PRECISION, input->
variable[i].precision);
00319     if (input->algorithm == ALGORITHM_SWEEP)
00320         xml_node_set_uint (child, XML_NSWEEPS, input->
variable[i].nsweeps);
00321     else if (input->algorithm == ALGORITHM_GENETIC)
00322         xml_node_set_uint (child, XML_NBITS, input->
variable[i].nbits);
00323     if (input->nsteps)
00324         xml_node_set_float (child, XML_STEP, input->
variable[i].step);
00325 }
00326
00327 // Saving the error norm
00328 switch (input->norm)
00329 {
00330     case ERROR_NORM_MAXIMUM:
00331         xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00332         break;
00333     case ERROR_NORM_P:
00334         xmlSetProp (node, XML_NORM, XML_P);
00335         xml_node_set_float (node, XML_P, input->p);
00336         break;
00337     case ERROR_NORM_TAXICAB:
00338         xmlSetProp (node, XML_NORM, XML_TAXICAB);
00339 }
00340
00341 // Saving the XML file
00342 xmlSaveFormatFile (filename, doc, 1);
00343
00344 // Freeing memory
00345 xmlFreeDoc (doc);
00346
00347 #if DEBUG
00348 fprintf (stderr, "input_save: end\n");
00349 #endif
00350 }

```

Here is the call graph for this function:



#### 5.11.2.2 void input\_save\_direction ( xmlNode \* node )

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

## Parameters

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

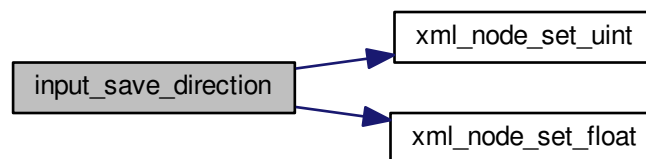
Definition at line 172 of file [interface.c](#).

```

00173 {
00174     #if DEBUG
00175     fprintf (stderr, "input_save_direction: start\n");
00176     #endif
00177     if (input->nsteps)
00178     {
00179         xml_node_set_uint (node, XML_NSTEPS, input->
nsteps);
00180         if (input->relaxation != DEFAULT_RELAXATION)
00181             xml_node_set_float (node, XML_RELAXATION, input->
relaxation);
00182         switch (input->direction)
00183         {
00184             case DIRECTION_METHOD_COORDINATES:
00185                 xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
00186                 break;
00187             default:
00188                 xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
00189                 xml_node_set_uint (node, XML_NESTIMATES, input->
nestimates);
00190         }
00191     }
00192     #if DEBUG
00193     fprintf (stderr, "input_save_direction: end\n");
00194     #endif
00195 }

```

Here is the call graph for this function:



### 5.11.2.3 unsigned int window\_get\_algorithm ( )

Function to get the stochastic algorithm number.

## Returns

Stochastic algorithm number.

Definition at line 461 of file [interface.c](#).

```

00462 {
00463     unsigned int i;
00464     #if DEBUG
00465     fprintf (stderr, "window_get_algorithm: start\n");
00466     #endif
00467     i = gtk_array_get_active (window->button_algorithm,
NALGORITHMS);
00468     #if DEBUG
00469     fprintf (stderr, "window_get_algorithm: %u\n", i);
00470     fprintf (stderr, "window_get_algorithm: end\n");
00471     #endif
00472     return i;
00473 }

```

Here is the call graph for this function:



#### 5.11.2.4 unsigned int window\_get\_direction ( )

Function to get the direction search method number.

##### Returns

Direction search method number.

Definition at line 481 of file [interface.c](#).

```

00482 {
00483     unsigned int i;
00484     #if DEBUG
00485         fprintf (stderr, "window_get_direction: start\n");
00486     #endif
00487     i = gtk_array_get_active (window->button_direction,
00488                             NDIRECTIONS);
00489     #if DEBUG
00489         fprintf (stderr, "window_get_direction: %u\n", i);
00490         fprintf (stderr, "window_get_direction: end\n");
00491     #endif
00492     return i;
00493 }
  
```

Here is the call graph for this function:



#### 5.11.2.5 unsigned int window\_get\_norm ( )

Function to get the norm method number.

##### Returns

Norm method number.

Definition at line 501 of file [interface.c](#).



```

00502 {
00503     unsigned int i;
00504     #if DEBUG
00505         fprintf (stderr, "window_get_norm: start\n");
00506     #endif
00507     i = gtk_array_get_active (window->button_norm,
NNORMS);
00508     #if DEBUG
00509         fprintf (stderr, "window_get_norm: %u\n", i);
00510         fprintf (stderr, "window_get_norm: end\n");
00511     #endif
00512     return i;
00513 }

```

Here is the call graph for this function:



#### 5.11.2.6 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 1560 of file [interface.c](#).

```

01561 {
01562     unsigned int i;
01563     char *buffer;
01564     #if DEBUG
01565         fprintf (stderr, "window_read: start\n");
01566     #endif
01567     // Reading new input file
01568     input_free ();
01569     if (!input_open (filename))
01570         return 0;
01571     // Setting GTK+ widgets data
01572     gtk_entry_set_text (window->entry_result, input->result);
01573     gtk_entry_set_text (window->entry_variables, input->
variables);
01574     buffer = g_build_filename (input->directory, input->simulator, NULL);
01575     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01576     g_free (buffer);
01577     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01578     if (input->evaluator)
01579     {
01580         buffer = g_build_filename (input->directory, input->evaluator, NULL);
01581         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01582         g_free (buffer);
01583     }
01584     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->

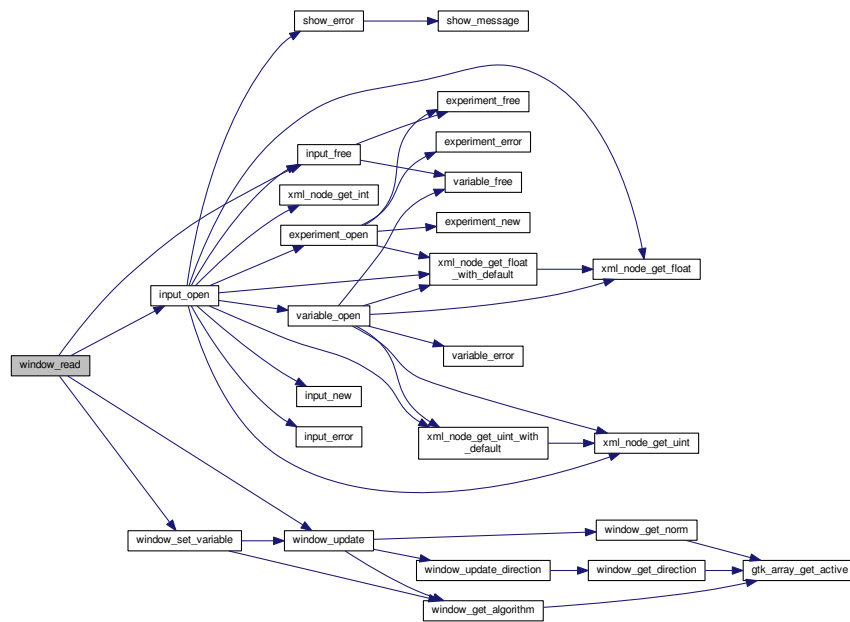
```

```

        algorithm)), TRUE);
01591     switch (input->algorithm)
01592     {
01593         case ALGORITHM_MONTE_CARLO:
01594             gtk_spin_button_set_value (window->spin_simulations,
01595                                     (gdouble) input->nsimulations);
01596         case ALGORITHM_SWEEP:
01597             gtk_spin_button_set_value (window->spin_iterations,
01598                                     (gdouble) input->niterations);
01599             gtk_spin_button_set_value (window->spin_best, (gdouble) input->
nbest);
01600             gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01601             gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_direction),
                                     input->nsteps);
01602             if (input->nsteps)
01603             {
01604                 gtk_toggle_button_set_active
01605                     (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01606                 gtk_spin_button_set_value (window->spin_steps,
01607                                     (gdouble) input->nsteps);
01608                 gtk_spin_button_set_value (window->spin_relaxation,
01609                                     (gdouble) input->relaxation);
01610                 switch (input->direction)
01611                 {
01612                     case DIRECTION_METHOD_RANDOM:
01613                         gtk_spin_button_set_value (window->spin_estimates,
01614                                     (gdouble) input->nestimates);
01615                     }
01616                 break;
01617             }
01618             default:
01619                 gtk_spin_button_set_value (window->spin_population,
01620                                     (gdouble) input->nsimulations);
01621                 gtk_spin_button_set_value (window->spin_generations,
01622                                     (gdouble) input->niterations);
01623                 gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01624                 gtk_spin_button_set_value (window->spin_reproduction,
01625                                     input->reproduction_ratio);
01626                 gtk_spin_button_set_value (window->spin_adaptation,
01627                                     input->adaptation_ratio);
01628             }
01629             gtk_toggle_button_set_active
01630                 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01631             gtk_spin_button_set_value (window->spin_p, input->p);
01632             gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01633             g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01634             g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
01635             gtk_combo_box_text_remove_all (window->combo_experiment);
01636             for (i = 0; i < input->nexperiments; ++i)
01637                 gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i].name);
01638             g_signal_handler_unblock
01639                 (window->button_experiment, window->
id_experiment_name);
01640             g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01641             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01642             g_signal_handler_block (window->combo_variable, window->
id_variable);
01643             g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01644             gtk_combo_box_text_remove_all (window->combo_variable);
01645             for (i = 0; i < input->nvariables; ++i)
01646                 gtk_combo_box_text_append_text (window->combo_variable,
input->variable[i].name);
01647             g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01648             g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01649             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01650             window_set_variable ();
01651             window_update ();
01652         }
01653     }
01654     #if DEBUG
01655     fprintf (stderr, "window_read: end\n");
01656     #endif
01657     return 1;
01658 }

```

Here is the call graph for this function:



### 5.11.2.7 int window\_save ( )

Function to save the input file.

#### Returns

1 on OK, 0 on Cancel.

Definition at line 554 of file [interface.c](#).

```

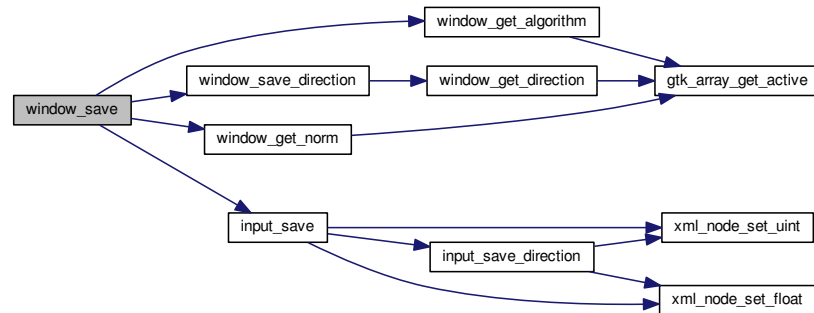
00555 {
00556     GtkFileChooserDialog *dlg;
00557     GtkFileFilter *filter;
00558     char *buffer;
00559
00560     #if DEBUG
00561         fprintf (stderr, "window_save: start\n");
00562     #endif
00563
00564     // Opening the saving dialog
00565     dlg = (GtkFileChooserDialog *)
00566         gtk_file_chooser_dialog_new (gettext ("Save file"),
00567                                     window->window,
00568                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00569                                     gettext ("_Cancel"),
00570                                     GTK_RESPONSE_CANCEL,
00571                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00572     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00573     buffer = g_build_filename (input->directory, input->name, NULL);
00574     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00575     g_free (buffer);
00576
00577     // Adding XML filter
00578     filter = (GtkFileFilter *) gtk_file_filter_new ();
00579     gtk_file_filter_set_name (filter, "XML");
00580     gtk_file_filter_add_pattern (filter, "*.xml");
00581     gtk_file_filter_add_pattern (filter, "*.XML");
00582     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00583
00584     // If OK response then saving
00585     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
  
```

```

00586     {
00587
00588         // Adding properties to the root XML node
00589         input->simulator = gtk_file_chooser_get_filename
00590             (GTK_FILE_CHOOSER (window->button_simulator));
00591         if (gtk_toggle_button_get_active
00592             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00593             input->evaluator = gtk_file_chooser_get_filename
00594                 (GTK_FILE_CHOOSER (window->button_evaluator));
00595         else
00596             input->evaluator = NULL;
00597         input->result
00598             = (char *) xmlStrdup ((const xmlChar *)
00599                 gtk_entry_get_text (window->entry_result));
00600         input->variables
00601             = (char *) xmlStrdup ((const xmlChar *)
00602                 gtk_entry_get_text (window->entry_variables));
00603
00604         // Setting the algorithm
00605         switch (window_get_algorithm ())
00606         {
00607             case ALGORITHM_MONTE_CARLO:
00608                 input->algorithm = ALGORITHM_MONTE_CARLO;
00609                 input->nsimulations
00610                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
00611                 input->niterations
00612                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00613                 input->tolerance = gtk_spin_button_get_value (window->
00614                     spin_tolerance);
00615                 input->nbest = gtk_spin_button_get_value_as_int (window->
00616                     spin_bests);
00617                 window_save_direction ();
00618                 break;
00619             case ALGORITHM_SWEEP:
00620                 input->algorithm = ALGORITHM_SWEEP;
00621                 input->niterations
00622                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00623                 input->tolerance = gtk_spin_button_get_value (window->
00624                     spin_tolerance);
00625                 input->nbest = gtk_spin_button_get_value_as_int (window->
00626                     spin_bests);
00627                 window_save_direction ();
00628                 break;
00629             default:
00630                 input->algorithm = ALGORITHM_GENETIC;
00631                 input->nsimulations
00632                     = gtk_spin_button_get_value_as_int (window->spin_population);
00633                 input->niterations
00634                     = gtk_spin_button_get_value_as_int (window->spin_generations);
00635                 input->mutation_ratio
00636                     = gtk_spin_button_get_value (window->spin_mutation);
00637                 input->reproduction_ratio
00638                     = gtk_spin_button_get_value (window->spin_reproduction);
00639                 input->adaptation_ratio
00640                     = gtk_spin_button_get_value (window->spin_adaptation);
00641                 break;
00642         }
00643         input->norm = window_get_norm ();
00644         input->p = gtk_spin_button_get_value (window->spin_p);
00645         input->threshold = gtk_spin_button_get_value (window->
00646             spin_threshold);
00647
00648         // Saving the XML file
00649         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00650         input_save (buffer);
00651
00652         // Closing and freeing memory
00653         g_free (buffer);
00654         gtk_widget_destroy (GTK_WIDGET (dlg));
00655 #if DEBUG
00656         fprintf (stderr, "window_save: end\n");
00657 #endif
00658         return 1;
00659     }
00660
00661     // Closing and freeing memory
00662     gtk_widget_destroy (GTK_WIDGET (dlg));
00663 #if DEBUG
00664     fprintf (stderr, "window_save: end\n");
00665 #endif
00666     return 0;
00667 }

```

Here is the call graph for this function:



#### 5.11.2.8 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 1210 of file [interface.c](#).

```

01211 {
01212     unsigned int i, j;
01213     char *buffer;
01214     GFile *file1, *file2;
01215     #if DEBUG
01216         fprintf (stderr, "window_template_experiment: start\n");
01217     #endif
01218     i = (size_t) data;
01219     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01220     file1
01221         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01222     file2 = g_file_new_for_path (input->directory);
01223     buffer = g_file_get_relative_path (file2, file1);
01224     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01225     g_free (buffer);
01226     g_object_unref (file2);
01227     g_object_unref (file1);
01228     #if DEBUG
01229         fprintf (stderr, "window_template_experiment: end\n");
01230     #endif
01231 }

```

## 5.12 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 Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the

```

```

00018         documentation and/or other materials provided with the distribution.
00019
00020     THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021     WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023     SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024     SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025     PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027     CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028     IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029     OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #ifdef G_OS_WIN32
00044 #include <windows.h>
00045 #elif !defined(BSD)
00046 #include <alloca.h>
00047 #endif
00048 #if HAVE_MPI
00049 #include <mpi.h>
00050 #endif
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
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 #include "interface.h"
00060
00061 #define DEBUG 0
00062
00063 #ifdef G_OS_WIN32
00064 #define INPUT_FILE "test-ga-win.xml"
00065 #else
00066 #define INPUT_FILE "test-ga.xml"
00067 #endif
00068
00069 const char *logo[] = {
00070     "32 32 3 1",
00071     "      c None",
00072     ".      c #0000FF",
00073 "+      c #FF0000",
00074 "      ",
00075 "      ",
00076 "      ",
00077 "      .      .      .      .      ",
00078 "      .      .      .      .      ",
00079 "      .      .      .      .      ",
00080 "      .      .      .      .      ",
00081 "      .      .      .      .      ",
00082 "      .      .      .      .      ",
00083 "      .      .      .      .      ",
00084 "      .      .      .      .      ",
00085 "      .      .      .      .      ",
00086 "      .      .      .      .      ",
00087 "      .      .      .      .      ",
00088 "      .      .      .      .      ",
00089 "      .      .      .      .      ",
00090 "      .      .      .      .      ",
00091 "      .      .      .      .      ",
00092 "      .      .      .      .      ",
00093 "      .      .      .      .      ",
00094 "      .      .      .      .      ",
00095 "      .      .      .      .      ",
00096 "      .      .      .      .      ",
00097 "      .      .      .      .      ",
00098 "      .      .      .      .      ",
00099 "      .      .      .      .      ",
00100 "      .      .      .      .      ",
00101 "      .      .      .      .      ",
00102 "      .      .      .      .      ",
00103 "      .      .      .      .      ",
00104 "      .      .      .      .      ",
00105 "      .      .      .      .      ",
00106 "      .      .      .      .      ",
00107 "      .      .      .      .      ",
00108 "      .      .      .      .      ",
00109 "      .      .      .      .      ",
00110 "      .      .      .      .      ",
00111 "      .      .      .      .      ",
00112 "      .      .      .      .      ",
00113 "      .      .      .      .      "

```

```

00114     "
00115     "
00116 };
00117
00118 /*
00119 const char * logo[] = {
00120 "32 32 3 1",
00121 "    c #FFFFFFFFFFFF",
00122 ".    c #00000000FFFF",
00123 "X    c #FFFF00000000",
00124 "
00125 "
00126 "
00127 "    .    .    .    .    "
00128 "    .    .    .    .    "
00129 "    .    .    .    .    "
00130 "    .    .    .    .    "
00131 "    .    .    XXX    .    "
00132 "    .    .    XXXXX   .    "
00133 "    .    .    XXXXX   .    "
00134 "    .    .    XXXXX   .    "
00135 "   XXX    .    XXX    XXX    "
00136 "  XXXXX   .    .    XXXXX   "
00137 "  XXXXX   .    .    XXXXX   "
00138 "  XXXXX   .    .    XXXXX   "
00139 "   XXX    .    .    XXX    "
00140 "    .    .    .    .    "
00141 "    .    XXX    .    .    "
00142 "    .    XXXXX   .    .    "
00143 "    .    XXXXX   .    .    "
00144 "    .    XXXXX   .    .    "
00145 "    .    XXX    .    .    "
00146 "    .    .    .    .    "
00147 "    .    .    .    .    "
00148 "    .    .    .    .    "
00149 "    .    .    .    .    "
00150 "    .    .    .    .    "
00151 "    .    .    .    .    "
00152 "    .    .    .    .    "
00153 "
00154 "
00155 "    "};
00156 */
00157
00158 Options options[1];
00160 Running running[1];
00162 Window window[1];
00164
00171 void
00172 input_save_direction (xmlNode * node)
00173 {
00174     #if DEBUG
00175     fprintf (stderr, "input_save_direction: start\n");
00176     #endif
00177     if (input->nsteps)
00178     {
00179         xml_node_set_uint (node, XML_NSTEPS, input->
nsteps);
00180         if (input->relaxation != DEFAULT_RELAXATION)
00181             xml_node_set_float (node, XML_RELAXATION, input->
relaxation);
00182         switch (input->direction)
00183         {
00184             case DIRECTION_METHOD_COORDINATES:
00185                 xmlSetProp (node, XML_DIRECTION, XML_COORDINATES);
00186                 break;
00187             default:
00188                 xmlSetProp (node, XML_DIRECTION, XML_RANDOM);
00189                 xml_node_set_uint (node, XML_NESTIMATES, input->
nestimates);
00190         }
00191     }
00192     #if DEBUG
00193     fprintf (stderr, "input_save_direction: end\n");
00194     #endif
00195 }
00196
00203 void
00204 input_save (char *filename)
00205 {
00206     unsigned int i, j;
00207     char *buffer;
00208     xmlDoc *doc;
00209     xmlNode *node, *child;
00210     GFile *file, *file2;
00211
00212     #if DEBUG

```

```

00213     fprintf (stderr, "input_save: start\n");
00214 #endif
00215
00216     // Getting the input file directory
00217     input->name = g_path_get_basename (filename);
00218     input->directory = g_path_get_dirname (filename);
00219     file = g_file_new_for_path (input->directory);
00220
00221     // Opening the input file
00222     doc = xmlNewDoc ((const xmlChar *) "1.0");
00223
00224     // Setting root XML node
00225     node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00226     xmlDocSetRootElement (doc, node);
00227
00228     // Adding properties to the root XML node
00229     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00230         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00231     if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
00232         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
00233     file2 = g_file_new_for_path (input->simulator);
00234     buffer = g_file_get_relative_path (file, file2);
00235     g_object_unref (file2);
00236     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00237     g_free (buffer);
00238     if (input->evaluator)
00239     {
00240         file2 = g_file_new_for_path (input->evaluator);
00241         buffer = g_file_get_relative_path (file, file2);
00242         g_object_unref (file2);
00243         if (xmlStrlen ((xmlChar *) buffer))
00244             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00245         g_free (buffer);
00246     }
00247     if (input->seed != DEFAULT_RANDOM_SEED)
00248         xml_node_set_uint (node, XML_SEED, input->seed);
00249
00250     // Setting the algorithm
00251     buffer = (char *) g_malloc (64);
00252     switch (input->algorithm)
00253     {
00254     case ALGORITHM_MONTE_CARLO:
00255         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00256         snprintf (buffer, 64, "%u", input->nsimulations);
00257         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00258         snprintf (buffer, 64, "%u", input->niterations);
00259         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00260         snprintf (buffer, 64, "%.3lg", input->tolerance);
00261         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00262         snprintf (buffer, 64, "%u", input->nbest);
00263         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00264         input_save_direction (node);
00265         break;
00266     case ALGORITHM_SWEEP:
00267         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00268         snprintf (buffer, 64, "%u", input->niterations);
00269         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00270         snprintf (buffer, 64, "%.3lg", input->tolerance);
00271         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00272         snprintf (buffer, 64, "%u", input->nbest);
00273         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00274         input_save_direction (node);
00275         break;
00276     default:
00277         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00278         snprintf (buffer, 64, "%u", input->nsimulations);
00279         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00280         snprintf (buffer, 64, "%u", input->niterations);
00281         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00282         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00283         xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00284         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00285         xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00286         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00287         xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00288         break;
00289     }
00290     g_free (buffer);
00291     if (input->threshold != 0.)
00292         xml_node_set_float (node, XML_THRESHOLD, input->
threshold);
00293
00294     // Setting the experimental data
00295     for (i = 0; i < input->nexperiments; ++i)
00296     {
00297         child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00298         xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i].

```



```

    name);
00299     if (input->experiment[i].weight != 1.)
00300         xml_node_set_float (child, XML_WEIGHT, input->
experiment[i].weight);
00301     for (j = 0; j < input->experiment->ninputs; ++j)
00302         xmlSetProp (child, template[j],
00303                     (xmlChar *) input->experiment[i].template[j]);
00304     }
00305
00306     // Setting the variables data
00307     for (i = 0; i < input->nvariables; ++i)
00308     {
00309         child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00310         xmlSetProp (child, XML_NAME, (xmlChar *) input->variable[i].
name);
00311         xml_node_set_float (child, XML_MINIMUM, input->
variable[i].rangemin);
00312         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00313             xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->variable[i].rangeminabs);
00314         xml_node_set_float (child, XML_MAXIMUM, input->
variable[i].rangemax);
00315         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00316             xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->variable[i].rangemaxabs);
00317         if (input->variable[i].precision != DEFAULT_PRECISION)
00318             xml_node_set_uint (child, XML_PRECISION, input->
variable[i].precision);
00319         if (input->algorithm == ALGORITHM_SWEEP)
00320             xml_node_set_uint (child, XML_NSWEEPS, input->
variable[i].nsweeps);
00321         else if (input->algorithm == ALGORITHM_GENETIC)
00322             xml_node_set_uint (child, XML_NBITS, input->
variable[i].nbits);
00323         if (input->nsteps)
00324             xml_node_set_float (child, XML_STEP, input->
variable[i].step);
00325     }
00326
00327     // Saving the error norm
00328     switch (input->norm)
00329     {
00330     case ERROR_NORM_MAXIMUM:
00331         xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00332         break;
00333     case ERROR_NORM_P:
00334         xmlSetProp (node, XML_NORM, XML_P);
00335         xml_node_set_float (node, XML_P, input->p);
00336         break;
00337     case ERROR_NORM_TAXICAB:
00338         xmlSetProp (node, XML_NORM, XML_TAXICAB);
00339     }
00340
00341     // Saving the XML file
00342     xmlSaveFormatFile (filename, doc, 1);
00343
00344     // Freeing memory
00345     xmlFreeDoc (doc);
00346
00347     #if DEBUG
00348     fprintf (stderr, "input_save: end\n");
00349     #endif
00350
00351     void
00352     options_new ()
00353     {
00354     #if DEBUG
00355     fprintf (stderr, "options_new: start\n");
00356     #endif
00357     options->label_seed = (GtkLabel *)
gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
00358     options->spin_seed = (GtkSpinButton *)
gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00359     gtk_widget_set_tooltip_text
(GTK_WIDGET (options->spin_seed),
00360     gettext ("Seed to init the pseudo-random numbers generator"));
00361     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
00362     options->label_threads = (GtkLabel *)
gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
00363     options->spin_threads
= (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00364     gtk_widget_set_tooltip_text
(GTK_WIDGET (options->spin_threads),
00365     gettext ("Number of threads to perform the calibration/optimization for "
00366     "the stochastic algorithm"));
00367     gtk_spin_button_set_value (options->spin_threads, (gdouble)

```

```

nthreads);
00381 options->label_direction = (GtkLabel *)
00382     gtk_label_new (gettext ("Threads number for the direction search method"));
00383 options->spin_direction
00384     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00385 gtk_widget_set_tooltip_text
00386     (GTK_WIDGET (options->spin_direction),
00387      gettext ("Number of threads to perform the calibration/optimization for "
00388              "the direction search method"));
00389 gtk_spin_button_set_value (options->spin_direction,
00390                           (gdouble) nthreads_direction);
00391 options->grid = (GtkGrid *) gtk_grid_new ();
00392 gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00393 gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00394 gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00395                 0, 1, 1, 1);
00396 gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00397                 1, 1, 1, 1);
00398 gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
00399                 0, 2, 1, 1);
00400 gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
00401                 1, 2, 1, 1);
00402 gtk_widget_show_all (GTK_WIDGET (options->grid));
00403 options->dialog = (GtkDialog *)
00404     gtk_dialog_new_with_buttons (gettext ("Options"),
00405                                 window->window,
00406                                 GTK_DIALOG_MODAL,
00407                                 gettext ("_OK"), GTK_RESPONSE_OK,
00408                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
00409                                 NULL);
00410 gtk_container_add
00411     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00412      GTK_WIDGET (options->grid));
00413 if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00414 {
00415     input->seed
00416         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00417     nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00418     nthreads_direction
00419         = gtk_spin_button_get_value_as_int (options->spin_direction);
00420 }
00421 gtk_widget_destroy (GTK_WIDGET (options->dialog));
00422 #if DEBUG
00423     fprintf (stderr, "options_new: end\n");
00424 #endif
00425 }
00426
00431 void
00432 running_new ()
00433 {
00434     #if DEBUG
00435         fprintf (stderr, "running_new: start\n");
00436     #endif
00437     running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
00438     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00439     running->grid = (GtkGrid *) gtk_grid_new ();
00440     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00441     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00442     running->dialog = (GtkDialog *)
00443         gtk_dialog_new_with_buttons (gettext ("Calculating"),
00444                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00445     gtk_container_add
00446         (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
00447          GTK_WIDGET (running->grid));
00448     gtk_spinner_start (running->spinner);
00449     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00450     #if DEBUG
00451         fprintf (stderr, "running_new: end\n");
00452     #endif
00453 }
00454
00460 unsigned int
00461 window_get_algorithm ()
00462 {
00463     unsigned int i;
00464     #if DEBUG
00465         fprintf (stderr, "window_get_algorithm: start\n");
00466     #endif
00467     i = gtk_array_get_active (window->button_algorithm,
00468                              NALGORITHMS);
00469     #if DEBUG
00470         fprintf (stderr, "window_get_algorithm: %u\n", i);
00471     #endif
00472     return i;
00473 }
00474

```

```

00480 unsigned int
00481 window_get_direction ()
00482 {
00483     unsigned int i;
00484     #if DEBUG
00485     fprintf (stderr, "window_get_direction: start\n");
00486     #endif
00487     i = gtk_array_get_active (window->button_direction,
00488                             NDIRECTIONS);
00489     #if DEBUG
00490     fprintf (stderr, "window_get_direction: %u\n", i);
00491     fprintf (stderr, "window_get_direction: end\n");
00492     #endif
00493     return i;
00494 }
00495
00500 unsigned int
00501 window_get_norm ()
00502 {
00503     unsigned int i;
00504     #if DEBUG
00505     fprintf (stderr, "window_get_norm: start\n");
00506     #endif
00507     i = gtk_array_get_active (window->button_norm,
00508                             NNORMS);
00509     #if DEBUG
00510     fprintf (stderr, "window_get_norm: %u\n", i);
00511     fprintf (stderr, "window_get_norm: end\n");
00512     #endif
00513     return i;
00514 }
00515
00519 void
00520 window_save_direction ()
00521 {
00522     #if DEBUG
00523     fprintf (stderr, "window_save_direction: start\n");
00524     #endif
00525     if (gtk_toggle_button_get_active
00526         (GTK_TOGGLE_BUTTON (window->check_direction)))
00527     {
00528         input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
00529         input->relaxation = gtk_spin_button_get_value (window->
00530 spin_relaxation);
00531         switch (window_get_direction ())
00532         {
00533             case DIRECTION_METHOD_COORDINATES:
00534                 input->direction = DIRECTION_METHOD_COORDINATES;
00535                 break;
00536             default:
00537                 input->direction = DIRECTION_METHOD_RANDOM;
00538                 input->nestimates
00539                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00540         }
00541     }
00542     else
00543         input->nsteps = 0;
00544     #if DEBUG
00545     fprintf (stderr, "window_save_direction: end\n");
00546     #endif
00547 }
00548
00553 int
00554 window_save ()
00555 {
00556     GtkFileChooserDialog *dlg;
00557     GtkFileFilter *filter;
00558     char *buffer;
00559
00560     #if DEBUG
00561     fprintf (stderr, "window_save: start\n");
00562     #endif
00563
00564     // Opening the saving dialog
00565     dlg = (GtkFileChooserDialog *)
00566         gtk_file_chooser_dialog_new (gettext ("Save file"),
00567                                     window->window,
00568                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00569                                     gettext ("_Cancel"),
00570                                     GTK_RESPONSE_CANCEL,
00571                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00572     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00573     buffer = g_build_filename (input->directory, input->name, NULL);
00574     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00575     g_free (buffer);
00576
00577     // Adding XML filter

```

```

00578 filter = (GtkFileFilter *) gtk_file_filter_new ();
00579 gtk_file_filter_set_name (filter, "XML");
00580 gtk_file_filter_add_pattern (filter, "*.xml");
00581 gtk_file_filter_add_pattern (filter, "*.XML");
00582 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00583
00584 // If OK response then saving
00585 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00586 {
00587     // Adding properties to the root XML node
00588     input->simulator = gtk_file_chooser_get_filename
00589         (GTK_FILE_CHOOSER (window->button_simulator));
00590     if (gtk_toggle_button_get_active
00591         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00592         input->evaluator = gtk_file_chooser_get_filename
00593             (GTK_FILE_CHOOSER (window->button_evaluator));
00594     else
00595         input->evaluator = NULL;
00596     input->result
00597         = (char *) xmlStrdup ((const xmlChar *)
00598             gtk_entry_get_text (window->entry_result));
00599     input->variables
00600         = (char *) xmlStrdup ((const xmlChar *)
00601             gtk_entry_get_text (window->entry_variables));
00602     // Setting the algorithm
00603     switch (window_get_algorithm ())
00604     {
00605     case ALGORITHM_MONTE_CARLO:
00606         input->algorithm = ALGORITHM_MONTE_CARLO;
00607         input->nsimulations
00608             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00609         input->niterations
00610             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00611         input->tolerance = gtk_spin_button_get_value (window->
00612             spin_tolerance);
00613         input->nbest = gtk_spin_button_get_value_as_int (window->
00614             spin_best);
00615         window_save_direction ();
00616         break;
00617     case ALGORITHM_SWEEP:
00618         input->algorithm = ALGORITHM_SWEEP;
00619         input->niterations
00620             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00621         input->tolerance = gtk_spin_button_get_value (window->
00622             spin_tolerance);
00623         input->nbest = gtk_spin_button_get_value_as_int (window->
00624             spin_best);
00625         window_save_direction ();
00626         break;
00627     default:
00628         input->algorithm = ALGORITHM_GENETIC;
00629         input->nsimulations
00630             = gtk_spin_button_get_value_as_int (window->spin_population);
00631         input->niterations
00632             = gtk_spin_button_get_value_as_int (window->spin_generations);
00633         input->mutation_ratio
00634             = gtk_spin_button_get_value (window->spin_mutation);
00635         input->reproduction_ratio
00636             = gtk_spin_button_get_value (window->spin_reproduction);
00637         input->adaptation_ratio
00638             = gtk_spin_button_get_value (window->spin_adaptation);
00639         break;
00640     }
00641     input->norm = window_get_norm ();
00642     input->p = gtk_spin_button_get_value (window->spin_p);
00643     input->threshold = gtk_spin_button_get_value (window->spin_threshold);
00644     // Saving the XML file
00645     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00646     input_save (buffer);
00647     // Closing and freeing memory
00648     g_free (buffer);
00649     gtk_widget_destroy (GTK_WIDGET (dlg));
00650 #if DEBUG
00651     fprintf (stderr, "window_save: end\n");
00652 #endif
00653     return 1;
00654 }
00655
00656 // Closing and freeing memory
00657 gtk_widget_destroy (GTK_WIDGET (dlg));
00658 #if DEBUG
00659     fprintf (stderr, "window_save: end\n");
00660 #endif

```

```

00661     return 0;
00662 }
00663
00664 void
00665 window_run ()
00666 {
00667     unsigned int i;
00668     char *msg, *msg2, buffer[64], buffer2[64];
00669     #if DEBUG
00670     fprintf (stderr, "window_run: start\n");
00671     #endif
00672     if (!window_save ())
00673     {
00674         #if DEBUG
00675         fprintf (stderr, "window_run: end\n");
00676         #endif
00677         return;
00678     }
00679     running_new ();
00680     while (gtk_events_pending ())
00681         gtk_main_iteration ();
00682     optimize_open ();
00683     #if DEBUG
00684     fprintf (stderr, "window_run: closing running dialog\n");
00685     #endif
00686     gtk_spinner_stop (running->spinner);
00687     gtk_widget_destroy (GTK_WIDGET (running->dialog));
00688     #if DEBUG
00689     fprintf (stderr, "window_run: displaying results\n");
00690     #endif
00691     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00692     msg2 = g_strdup (buffer);
00693     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00694     {
00695         snprintf (buffer, 64, "%s = %s\n",
00696                 input->variable[i].name, format[input->
00697                 variable[i].precision]);
00698         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
00699         msg2 = g_strconcat (msg2, buffer2, NULL);
00700         g_free (msg2);
00701     }
00702     snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
00703             optimize->calculation_time);
00704     msg2 = g_strconcat (msg2, buffer, NULL);
00705     g_free (msg2);
00706     show_message (gettext ("Best result"), msg, INFO_TYPE);
00707     g_free (msg);
00708     #if DEBUG
00709     fprintf (stderr, "window_run: freeing memory\n");
00710     #endif
00711     optimize_free ();
00712     #if DEBUG
00713     fprintf (stderr, "window_run: end\n");
00714     #endif
00715 }
00716
00717 void
00718 window_help ()
00719 {
00720     char *buffer, *buffer2;
00721     #if DEBUG
00722     fprintf (stderr, "window_help: start\n");
00723     #endif
00724     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
00725                               gettext ("user-manual.pdf"), NULL);
00726     buffer = g_filename_to_uri (buffer2, NULL, NULL);
00727     g_free (buffer2);
00728     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
00729     #if DEBUG
00730     fprintf (stderr, "window_help: uri=%s\n", buffer);
00731     #endif
00732     g_free (buffer);
00733     #if DEBUG
00734     fprintf (stderr, "window_help: end\n");
00735     #endif
00736 }
00737
00738 void
00739 window_about ()
00740 {
00741     static const gchar *authors[] = {
00742         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
00743         "Borja Latorre Garcés <borja.latorre@csic.es>",
00744         NULL
00745     };
00746     #if DEBUG
00747     fprintf (stderr, "window_about: start\n");
00748     #endif

```

```

00759 #endif
00760 gtk_show_about_dialog
00761     (window->window,
00762      "program_name", "MPCOTool",
00763      "comments",
00764      gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
00765              "A software to perform calibrations or optimizations of "
00766              "empirical parameters"),
00767      "authors", authors,
00768      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
00769      "version", "2.3.0",
00770      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
00771      "logo", window->logo,
00772      "website", "https://github.com/jburguete/mpcotool",
00773      "license-type", GTK_LICENSE_BSD, NULL);
00774 #if DEBUG
00775     fprintf (stderr, "window_about: end\n");
00776 #endif
00777 }
00778
00784 void
00785 window_update_direction ()
00786 {
00787     #if DEBUG
00788         fprintf (stderr, "window_update_direction: start\n");
00789     #endif
00790     gtk_widget_show (GTK_WIDGET (window->check_direction));
00791     if (gtk_toggle_button_get_active
00792         (GTK_TOGGLE_BUTTON (window->check_direction)))
00793     {
00794         gtk_widget_show (GTK_WIDGET (window->grid_direction));
00795         gtk_widget_show (GTK_WIDGET (window->label_step));
00796         gtk_widget_show (GTK_WIDGET (window->spin_step));
00797     }
00798     switch (window_get_direction ())
00799     {
00800     case DIRECTION_METHOD_COORDINATES:
00801         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
00802         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
00803         break;
00804     default:
00805         gtk_widget_show (GTK_WIDGET (window->label_estimates));
00806         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
00807     }
00808     #if DEBUG
00809         fprintf (stderr, "window_update_direction: end\n");
00810     #endif
00811 }
00812
00817 void
00818 window_update ()
00819 {
00820     unsigned int i;
00821     #if DEBUG
00822         fprintf (stderr, "window_update: start\n");
00823     #endif
00824     gtk_widget_set_sensitive
00825         (GTK_WIDGET (window->button_evaluator),
00826         gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
00827         (window->check_evaluator)));
00828     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
00829     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
00830     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
00831     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
00832     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
00833     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
00834     gtk_widget_hide (GTK_WIDGET (window->label_bests));
00835     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
00836     gtk_widget_hide (GTK_WIDGET (window->label_population));
00837     gtk_widget_hide (GTK_WIDGET (window->spin_population));
00838     gtk_widget_hide (GTK_WIDGET (window->label_generations));
00839     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
00840     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
00841     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
00842     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
00843     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
00844     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
00845     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
00846     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
00847     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
00848     gtk_widget_hide (GTK_WIDGET (window->label_bits));
00849     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
00850     gtk_widget_hide (GTK_WIDGET (window->check_direction));
00851     gtk_widget_hide (GTK_WIDGET (window->grid_direction));
00852     gtk_widget_hide (GTK_WIDGET (window->label_step));
00853     gtk_widget_hide (GTK_WIDGET (window->spin_step));
00854     gtk_widget_hide (GTK_WIDGET (window->label_p));

```

```

00855     gtk_widget_hide (GTK_WIDGET (window->spin_p));
00856     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
00857     switch (window_get_algorithm ())
00858     {
00859     case ALGORITHM_MONTE_CARLO:
00860         gtk_widget_show (GTK_WIDGET (window->label_simulations));
00861         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
00862         gtk_widget_show (GTK_WIDGET (window->label_iterations));
00863         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
00864         if (i > 1)
00865         {
00866             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
00867             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
00868             gtk_widget_show (GTK_WIDGET (window->label_bests));
00869             gtk_widget_show (GTK_WIDGET (window->spin_bests));
00870         }
00871         window_update_direction ();
00872         break;
00873     case ALGORITHM_SWEEP:
00874         gtk_widget_show (GTK_WIDGET (window->label_iterations));
00875         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
00876         if (i > 1)
00877         {
00878             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
00879             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
00880             gtk_widget_show (GTK_WIDGET (window->label_bests));
00881             gtk_widget_show (GTK_WIDGET (window->spin_bests));
00882         }
00883         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
00884         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
00885         gtk_widget_show (GTK_WIDGET (window->check_direction));
00886         window_update_direction ();
00887         break;
00888     default:
00889         gtk_widget_show (GTK_WIDGET (window->label_population));
00890         gtk_widget_show (GTK_WIDGET (window->spin_population));
00891         gtk_widget_show (GTK_WIDGET (window->label_generations));
00892         gtk_widget_show (GTK_WIDGET (window->spin_generations));
00893         gtk_widget_show (GTK_WIDGET (window->label_mutation));
00894         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
00895         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
00896         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
00897         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
00898         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
00899         gtk_widget_show (GTK_WIDGET (window->label_bits));
00900         gtk_widget_show (GTK_WIDGET (window->spin_bits));
00901     }
00902     gtk_widget_set_sensitive
00903     (GTK_WIDGET (window->button_remove_experiment), input->
n experiments > 1);
00904     gtk_widget_set_sensitive
00905     (GTK_WIDGET (window->button_remove_variable), input->
n variables > 1);
00906     for (i = 0; i < input->experiment->ninputs; ++i)
00907     {
00908         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
00909         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
00910         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
00911         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
00912         g_signal_handler_block
00913         (window->check_template[i], window->id_template[i]);
00914         g_signal_handler_block (window->button_template[i], window->
id_input[i]);
00915         gtk_toggle_button_set_active
00916         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
00917         g_signal_handler_unblock
00918         (window->button_template[i], window->id_input[i]);
00919         g_signal_handler_unblock
00920         (window->check_template[i], window->id_template[i]);
00921     }
00922     if (i > 0)
00923     {
00924         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
00925         gtk_widget_set_sensitive
00926         (GTK_WIDGET (window->button_template[i - 1]),
00927          gtk_toggle_button_get_active
00928          (GTK_TOGGLE_BUTTON (window->check_template[i - 1])));
00929     }
00930     if (i < MAX_NINPUTS)
00931     {
00932         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
00933         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
00934         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
00935         gtk_widget_set_sensitive
00936         (GTK_WIDGET (window->button_template[i]),
00937          gtk_toggle_button_get_active
00938          (GTK_TOGGLE_BUTTON (window->check_template[i])));

```

```

00939     g_signal_handler_block
00940     (window->check_template[i], window->id_template[i]);
00941     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
00942     gtk_toggle_button_set_active
00943     (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
00944     g_signal_handler_unblock
00945     (window->button_template[i], window->id_input[i]);
00946     g_signal_handler_unblock
00947     (window->check_template[i], window->id_template[i]);
00948 }
00949 while (++i < MAX_NINPUTS)
00950 {
00951     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
00952     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
00953 }
00954 gtk_widget_set_sensitive
00955 (GTK_WIDGET (window->spin_minabs),
00956  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
00957 gtk_widget_set_sensitive
00958 (GTK_WIDGET (window->spin_maxabs),
00959  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
00960 if (window_get_norm () == ERROR_NORM_P)
00961 {
00962     gtk_widget_show (GTK_WIDGET (window->label_p));
00963     gtk_widget_show (GTK_WIDGET (window->spin_p));
00964 }
00965 #if DEBUG
00966 fprintf (stderr, "window_update: end\n");
00967 #endif
00968 }
00969 void
00970 window_set_algorithm ()
00971 {
00972     int i;
00973     #if DEBUG
00974     fprintf (stderr, "window_set_algorithm: start\n");
00975     #endif
00976     i = window_get_algorithm ();
00977     switch (i)
00978     {
00979     case ALGORITHM_SWEEP:
00980         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
00981         if (i < 0)
00982             i = 0;
00983         gtk_spin_button_set_value (window->spin_sweeps,
(gdouble) input->variable[i].nsweeps);
00984         break;
00985     case ALGORITHM_GENETIC:
00986         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
00987         if (i < 0)
00988             i = 0;
00989         gtk_spin_button_set_value (window->spin_bits,
(gdouble) input->variable[i].nbits);
00990     }
00991     window_update ();
00992     #if DEBUG
00993     fprintf (stderr, "window_set_algorithm: end\n");
00994     #endif
00995 }
00996 void
00997 window_set_experiment ()
00998 {
00999     unsigned int i, j;
01000     char *buffer1, *buffer2;
01001     #if DEBUG
01002     fprintf (stderr, "window_set_experiment: start\n");
01003     #endif
01004     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01005     gtk_spin_button_set_value (window->spin_weight, input->experiment[i].
weight);
01006     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01007     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01008     g_free (buffer1);
01009     g_signal_handler_block
01010     (window->button_experiment, window->id_experiment_name);
01011     gtk_file_chooser_set_filename
01012     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01013     g_signal_handler_unblock
01014     (window->button_experiment, window->id_experiment_name);
01015     g_free (buffer2);
01016     for (j = 0; j < input->experiment->ninputs; ++j)
01017     {
01018         g_signal_handler_block (window->button_template[j], window->
id_input[j]);

```



```

01031     buffer2 = g_build_filename (input->directory,
01032                                input->experiment[i].template[j], NULL);
01033     gtk_file_chooser_set_filename
01034     (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01035     g_free (buffer2);
01036     g_signal_handler_unblock
01037     (window->button_template[j], window->id_input[j]);
01038 }
01039 #if DEBUG
01040 fprintf (stderr, "window_set_experiment: end\n");
01041 #endif
01042 }
01043
01044 void
01045 window_remove_experiment ()
01046 {
01047     unsigned int i, j;
01048     #if DEBUG
01049     fprintf (stderr, "window_remove_experiment: start\n");
01050     #endif
01051     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01052     g_signal_handler_block (window->combo_experiment, window->
01053                             id_experiment);
01054     gtk_combo_box_text_remove (window->combo_experiment, i);
01055     g_signal_handler_unblock (window->combo_experiment, window->
01056                              id_experiment);
01057     experiment_free (input->experiment + i);
01058     --input->nexperiments;
01059     for (j = i; j < input->nexperiments; ++j)
01060         memcpy (input->experiment + j, input->experiment + j + 1,
01061                sizeof (Experiment));
01062     j = input->nexperiments - 1;
01063     if (i > j)
01064         i = j;
01065     for (j = 0; j < input->experiment->ninputs; ++j)
01066         g_signal_handler_block (window->button_template[j], window->
01067                                 id_input[j]);
01068     g_signal_handler_block
01069     (window->button_experiment, window->id_experiment_name);
01070     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01071     g_signal_handler_unblock
01072     (window->button_experiment, window->id_experiment_name);
01073     for (j = 0; j < input->experiment->ninputs; ++j)
01074         g_signal_handler_unblock (window->button_template[j], window->
01075                                 id_input[j]);
01076     window_update ();
01077     #if DEBUG
01078     fprintf (stderr, "window_remove_experiment: end\n");
01079     #endif
01080 }
01081
01082 void
01083 window_add_experiment ()
01084 {
01085     unsigned int i, j;
01086     #if DEBUG
01087     fprintf (stderr, "window_add_experiment: start\n");
01088     #endif
01089     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01090     g_signal_handler_block (window->combo_experiment, window->
01091                             id_experiment);
01092     gtk_combo_box_text_insert_text
01093     (window->combo_experiment, i, input->experiment[i].
01094      name);
01095     g_signal_handler_unblock (window->combo_experiment, window->
01096                              id_experiment);
01097     input->experiment = (Experiment *) g_realloc
01098     (input->experiment, (input->nexperiments + 1) * sizeof (
01099      Experiment));
01100     for (j = input->nexperiments - 1; j > i; --j)
01101         memcpy (input->experiment + j + 1, input->experiment + j,
01102                sizeof (Experiment));
01103     input->experiment[j + 1].name
01104     = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
01105     input->experiment[j + 1].weight = input->experiment[j].
01106     weight;
01107     input->experiment[j + 1].ninputs = input->experiment[j].
01108     ninputs;
01109     for (j = 0; j < input->experiment->ninputs; ++j)
01110         input->experiment[i + 1].template[j]
01111         = (char *) xmlStrdup ((xmlChar *) input->experiment[i].template[j]);
01112     ++input->nexperiments;
01113     for (j = 0; j < input->experiment->ninputs; ++j)
01114         g_signal_handler_block (window->button_template[j], window->
01115                                 id_input[j]);
01116     g_signal_handler_block
01117     (window->button_experiment, window->id_experiment_name);

```

```

01115     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01116     g_signal_handler_unblock
01117     (window->button_experiment, window->id_experiment_name);
01118     for (j = 0; j < input->experiment->ninputs; ++j)
01119         g_signal_handler_unblock (window->button_template[j], window->
01120             id_input[j]);
01121     window_update ();
01122     #if DEBUG
01123     fprintf (stderr, "window_add_experiment: end\n");
01124     #endif
01125 }
01126
01127 void
01128 window_name_experiment ()
01129 {
01130     unsigned int i;
01131     char *buffer;
01132     GFile *file1, *file2;
01133     #if DEBUG
01134     fprintf (stderr, "window_name_experiment: start\n");
01135     #endif
01136     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01137     file1
01138     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01139     file2 = g_file_new_for_path (input->directory);
01140     buffer = g_file_get_relative_path (file2, file1);
01141     g_signal_handler_block (window->combo_experiment, window->
01142         id_experiment);
01143     gtk_combo_box_text_remove (window->combo_experiment, i);
01144     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01145     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01146     g_signal_handler_unblock (window->combo_experiment, window->
01147         id_experiment);
01148     g_free (buffer);
01149     g_object_unref (file2);
01150     g_object_unref (file1);
01151     #if DEBUG
01152     fprintf (stderr, "window_name_experiment: end\n");
01153     #endif
01154 }
01155
01156 void
01157 window_weight_experiment ()
01158 {
01159     unsigned int i;
01160     #if DEBUG
01161     fprintf (stderr, "window_weight_experiment: start\n");
01162     #endif
01163     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01164     input->experiment[i].weight = gtk_spin_button_get_value (window->
01165         spin_weight);
01166     #if DEBUG
01167     fprintf (stderr, "window_weight_experiment: end\n");
01168     #endif
01169 }
01170
01171 void
01172 window_inputs_experiment ()
01173 {
01174     unsigned int j;
01175     #if DEBUG
01176     fprintf (stderr, "window_inputs_experiment: start\n");
01177     #endif
01178     j = input->experiment->ninputs - 1;
01179     if (j
01180         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01181             (window->check_template[j])))
01182         --input->experiment->ninputs;
01183     if (input->experiment->ninputs < MAX_NINPUTS
01184         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01185             (window->check_template[j])))
01186         ++input->experiment->ninputs;
01187     window_update ();
01188     #if DEBUG
01189     fprintf (stderr, "window_inputs_experiment: end\n");
01190     #endif
01191 }
01192
01193 void
01194 window_template_experiment (void *data)
01195 {
01196     unsigned int i, j;
01197     char *buffer;
01198     GFile *file1, *file2;
01199     #if DEBUG
01200     fprintf (stderr, "window_template_experiment: start\n");
01201     #endif

```

```

01218     i = (size_t) data;
01219     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01220     file1
01221     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01222     file2 = g_file_new_for_path (input->directory);
01223     buffer = g_file_get_relative_path (file2, file1);
01224     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01225     g_free (buffer);
01226     g_object_unref (file2);
01227     g_object_unref (file1);
01228     #if DEBUG
01229     fprintf (stderr, "window_template_experiment: end\n");
01230     #endif
01231 }
01232
01237 void
01238 window_set_variable ()
01239 {
01240     unsigned int i;
01241     #if DEBUG
01242     fprintf (stderr, "window_set_variable: start\n");
01243     #endif
01244     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01245     g_signal_handler_block (window->entry_variable, window->
01246     id_variable_label);
01247     gtk_entry_set_text (window->entry_variable, input->variable[i].
01248     name);
01249     g_signal_handler_unblock (window->entry_variable, window->
01250     id_variable_label);
01251     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01252     rangemin);
01253     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01254     rangemax);
01255     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01256     {
01257         gtk_spin_button_set_value (window->spin_minabs,
01258         input->variable[i].rangeminabs);
01259         gtk_toggle_button_set_active
01260         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01261     }
01262     else
01263     {
01264         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01265         gtk_toggle_button_set_active
01266         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01267     }
01268     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01269     {
01270         gtk_spin_button_set_value (window->spin_maxabs,
01271         input->variable[i].rangemaxabs);
01272         gtk_toggle_button_set_active
01273         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01274     }
01275     else
01276     {
01277         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01278         gtk_toggle_button_set_active
01279         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01280     }
01281     gtk_spin_button_set_value (window->spin_precision,
01282     input->variable[i].precision);
01283     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01284     nsteps);
01285     if (input->nsteps)
01286     gtk_spin_button_set_value (window->spin_step, input->variable[i].
01287     step);
01288     #if DEBUG
01289     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01290     input->variable[i].precision);
01291     #endif
01292     switch (window_get_algorithm ())
01293     {
01294     case ALGORITHM_SWEEP:
01295         gtk_spin_button_set_value (window->spin_sweeps,
01296         (gdouble) input->variable[i].nsweeps);
01297         #if DEBUG
01298         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01299         input->variable[i].nsweeps);
01300         #endif
01301         break;
01302     case ALGORITHM_GENETIC:
01303         gtk_spin_button_set_value (window->spin_bits,
01304         (gdouble) input->variable[i].nbits);
01305         #if DEBUG
01306         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01307         input->variable[i].nbits);
01308         #endif
01309     }
01310 }

```

```

01302         break;
01303     }
01304     window_update ();
01305     #if DEBUG
01306     fprintf (stderr, "window_set_variable: end\n");
01307     #endif
01308 }
01309
01314 void
01315 window_remove_variable ()
01316 {
01317     unsigned int i, j;
01318     #if DEBUG
01319     fprintf (stderr, "window_remove_variable: start\n");
01320     #endif
01321     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01322     g_signal_handler_block (window->combo_variable, window->
01323         id_variable);
01324     gtk_combo_box_text_remove (window->combo_variable, i);
01325     g_signal_handler_unblock (window->combo_variable, window->
01326         id_variable);
01327     xmlFree (input->variable[i].name);
01328     --input->nvariables;
01329     for (j = i; j < input->nvariables; ++j)
01330         memcpy (input->variable + j, input->variable + j + 1, sizeof (
01331             Variable));
01332     j = input->nvariables - 1;
01333     if (i > j)
01334         i = j;
01335     g_signal_handler_block (window->entry_variable, window->
01336         id_variable_label);
01337     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01338     g_signal_handler_unblock (window->entry_variable, window->
01339         id_variable_label);
01340     window_update ();
01341     #if DEBUG
01342     fprintf (stderr, "window_remove_variable: end\n");
01343     #endif
01344 }
01345
01346 void
01347 window_add_variable ()
01348 {
01349     unsigned int i, j;
01350     #if DEBUG
01351     fprintf (stderr, "window_add_variable: start\n");
01352     #endif
01353     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01354     g_signal_handler_block (window->combo_variable, window->
01355         id_variable);
01356     gtk_combo_box_text_insert_text (window->combo_variable, i,
01357         input->variable[i].name);
01358     g_signal_handler_unblock (window->combo_variable, window->
01359         id_variable);
01360     input->variable = (Variable *) g_realloc
01361         (input->variable, (input->nvariables + 1) * sizeof (
01362             Variable));
01363     for (j = input->nvariables - 1; j > i; --j)
01364         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01365             Variable));
01366     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01367             Variable));
01368     input->variable[j + 1].name
01369         = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01370     ++input->nvariables;
01371     g_signal_handler_block (window->entry_variable, window->
01372         id_variable_label);
01373     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01374     g_signal_handler_unblock (window->entry_variable, window->
01375         id_variable_label);
01376     window_update ();
01377     #if DEBUG
01378     fprintf (stderr, "window_add_variable: end\n");
01379     #endif
01380 }
01381
01382 void
01383 window_label_variable ()
01384 {
01385     unsigned int i;
01386     const char *buffer;
01387     #if DEBUG
01388     fprintf (stderr, "window_label_variable: start\n");
01389     #endif
01390     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01391     buffer = gtk_entry_get_text (window->entry_variable);
01392     g_signal_handler_block (window->combo_variable, window->

```

```

        id_variable);
01389     gtk_combo_box_text_remove (window->combo_variable, i);
01390     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01391     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01392     g_signal_handler_unblock (window->combo_variable, window->
        id_variable);
01393     #if DEBUG
01394     fprintf (stderr, "window_label_variable: end\n");
01395     #endif
01396 }
01397
01402 void
01403 window_precision_variable ()
01404 {
01405     unsigned int i;
01406     #if DEBUG
01407     fprintf (stderr, "window_precision_variable: start\n");
01408     #endif
01409     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01410     input->variable[i].precision
01411     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01412     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
        precision);
01413     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
        precision);
01414     gtk_spin_button_set_digits (window->spin_minabs,
        input->variable[i].precision);
01415     gtk_spin_button_set_digits (window->spin_maxabs,
        input->variable[i].precision);
01416     #if DEBUG
01417     fprintf (stderr, "window_precision_variable: end\n");
01418     #endif
01419 }
01420
01427 void
01428 window_rangemin_variable ()
01429 {
01430     unsigned int i;
01431     #if DEBUG
01432     fprintf (stderr, "window_rangemin_variable: start\n");
01433     #endif
01434     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01435     input->variable[i].rangemin = gtk_spin_button_get_value (window->
        spin_min);
01436     #if DEBUG
01437     fprintf (stderr, "window_rangemin_variable: end\n");
01438     #endif
01439 }
01440
01445 void
01446 window_rangemax_variable ()
01447 {
01448     unsigned int i;
01449     #if DEBUG
01450     fprintf (stderr, "window_rangemax_variable: start\n");
01451     #endif
01452     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01453     input->variable[i].rangemax = gtk_spin_button_get_value (window->
        spin_max);
01454     #if DEBUG
01455     fprintf (stderr, "window_rangemax_variable: end\n");
01456     #endif
01457 }
01458
01463 void
01464 window_rangeminabs_variable ()
01465 {
01466     unsigned int i;
01467     #if DEBUG
01468     fprintf (stderr, "window_rangeminabs_variable: start\n");
01469     #endif
01470     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01471     input->variable[i].rangeminabs
01472     = gtk_spin_button_get_value (window->spin_minabs);
01473     #if DEBUG
01474     fprintf (stderr, "window_rangeminabs_variable: end\n");
01475     #endif
01476 }
01477
01482 void
01483 window_rangemaxabs_variable ()
01484 {
01485     unsigned int i;
01486     #if DEBUG
01487     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01488     #endif
01489     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));

```

```

01490     input->variable[i].rangemaxabs
01491     = gtk_spin_button_get_value (window->spin_maxabs);
01492     #if DEBUG
01493     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01494     #endif
01495 }
01496
01501 void
01502 window_step_variable ()
01503 {
01504     unsigned int i;
01505     #if DEBUG
01506     fprintf (stderr, "window_step_variable: start\n");
01507     #endif
01508     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01509     input->variable[i].step = gtk_spin_button_get_value (window->
01510     spin_step);
01511     #if DEBUG
01512     fprintf (stderr, "window_step_variable: end\n");
01513     #endif
01514 }
01515
01519 void
01520 window_update_variable ()
01521 {
01522     int i;
01523     #if DEBUG
01524     fprintf (stderr, "window_update_variable: start\n");
01525     #endif
01526     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01527     if (i < 0)
01528         i = 0;
01529     switch (window_get_algorithm ())
01530     {
01531         case ALGORITHM_SWEEP:
01532             input->variable[i].nsweeps
01533             = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01534             #if DEBUG
01535             fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01536             input->variable[i].nsweeps);
01537             #endif
01538             break;
01539         case ALGORITHM_GENETIC:
01540             input->variable[i].nbits
01541             = gtk_spin_button_get_value_as_int (window->spin_bits);
01542             #if DEBUG
01543             fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01544             input->variable[i].nbits);
01545             #endif
01546         }
01547     #if DEBUG
01548     fprintf (stderr, "window_update_variable: end\n");
01549     #endif
01550 }
01551
01559 int
01560 window_read (char *filename)
01561 {
01562     unsigned int i;
01563     char *buffer;
01564     #if DEBUG
01565     fprintf (stderr, "window_read: start\n");
01566     #endif
01567
01568     // Reading new input file
01569     input_free ();
01570     if (!input_open (filename))
01571         return 0;
01572
01573     // Setting GTK+ widgets data
01574     gtk_entry_set_text (window->entry_result, input->result);
01575     gtk_entry_set_text (window->entry_variables, input->variables);
01576     buffer = g_build_filename (input->directory, input->simulator, NULL);
01577     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01578     (window->button_simulator), buffer);
01579     g_free (buffer);
01580     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01581     (size_t) input->evaluator);
01582     if (input->evaluator)
01583     {
01584         buffer = g_build_filename (input->directory, input->evaluator, NULL);
01585         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01586         (window->button_evaluator), buffer);
01587         g_free (buffer);
01588     }
01589     gtk_toggle_button_set_active
01590     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->

```

```

        algorithm]), TRUE);
01591     switch (input->algorithm)
01592     {
01593         case ALGORITHM_MONTE_CARLO:
01594             gtk_spin_button_set_value (window->spin_simulations,
01595                                         (gdouble) input->nsimulations);
01596         case ALGORITHM_SWEEP:
01597             gtk_spin_button_set_value (window->spin_iterations,
01598                                         (gdouble) input->niterations);
01599             gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01600             gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01601             gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
01602                                             input->nsteps);
01603             if (input->nsteps)
01604             {
01605                 gtk_toggle_button_set_active
01606                     (GTK_TOGGLE_BUTTON (window->button_direction
01607                                             [input->direction]), TRUE);
01608                 gtk_spin_button_set_value (window->spin_steps,
01609                                             (gdouble) input->nsteps);
01610                 gtk_spin_button_set_value (window->spin_relaxation,
01611                                             (gdouble) input->relaxation);
01612                 switch (input->direction)
01613                 {
01614                     case DIRECTION_METHOD_RANDOM:
01615                         gtk_spin_button_set_value (window->spin_estimates,
01616                                                     (gdouble) input->nestimates);
01617                 }
01618             }
01619             break;
01620         default:
01621             gtk_spin_button_set_value (window->spin_population,
01622                                         (gdouble) input->nsimulations);
01623             gtk_spin_button_set_value (window->spin_generations,
01624                                         (gdouble) input->niterations);
01625             gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01626             gtk_spin_button_set_value (window->spin_reproduction,
01627                                         input->reproduction_ratio);
01628             gtk_spin_button_set_value (window->spin_adaptation,
01629                                         input->adaptation_ratio);
01630             }
01631             gtk_toggle_button_set_active
01632                 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01633             gtk_spin_button_set_value (window->spin_p, input->p);
01634             gtk_spin_button_set_value (window->spin_threshold, input->threshold);
01635             g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01636             g_signal_handler_block (window->button_experiment,
01637                                     window->id_experiment_name);
01638             gtk_combo_box_text_remove_all (window->combo_experiment);
01639             for (i = 0; i < input->nexperiments; ++i)
01640                 gtk_combo_box_text_append_text (window->combo_experiment,
01641                                                 input->experiment[i].name);
01642             g_signal_handler_unblock
01643                 (window->button_experiment, window->id_experiment_name);
01644             g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01645             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01646             g_signal_handler_block (window->combo_variable, window->
id_variable);
01647             g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01648             gtk_combo_box_text_remove_all (window->combo_variable);
01649             for (i = 0; i < input->nvariables; ++i)
01650                 gtk_combo_box_text_append_text (window->combo_variable,
01651                                                 input->variable[i].name);
01652             g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01653             g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01654             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01655             window_set_variable ();
01656             window_update ();
01657
01658 #if DEBUG
01659     fprintf (stderr, "window_read: end\n");
01660 #endif
01661     return 1;
01662 }
01663
01664 void
01665 window_open ()
01666 {
01667     GtkFileChooserDialog *dlg;

```

```

01672  GtkFileFilter *filter;
01673  char *buffer, *directory, *name;
01674
01675  #if DEBUG
01676  fprintf (stderr, "window_open: start\n");
01677  #endif
01678
01679  // Saving a backup of the current input file
01680  directory = g_strdup (input->directory);
01681  name = g_strdup (input->name);
01682
01683  // Opening dialog
01684  dlg = (GtkFileChooserDialog *)
01685  gtk_file_chooser_dialog_new (gettext ("Open input file"),
01686  window->window,
01687  GTK_FILE_CHOOSER_ACTION_OPEN,
01688  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
01689  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
01690
01691  // Adding XML filter
01692  filter = (GtkFileFilter *) gtk_file_filter_new ();
01693  gtk_file_filter_set_name (filter, "XML");
01694  gtk_file_filter_add_pattern (filter, "*.xml");
01695  gtk_file_filter_add_pattern (filter, "*.XML");
01696  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
01697
01698  // If OK saving
01699  while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
01700  {
01701
01702      // Traying to open the input file
01703      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
01704      if (!window_read (buffer))
01705      {
01706  #if DEBUG
01707      fprintf (stderr, "window_open: error reading input file\n");
01708  #endif
01709      g_free (buffer);
01710
01711      // Reading backup file on error
01712      buffer = g_build_filename (directory, name, NULL);
01713      if (!input_open (buffer))
01714      {
01715
01716          // Closing on backup file reading error
01717  #if DEBUG
01718      fprintf (stderr, "window_read: error reading backup file\n");
01719  #endif
01720      g_free (buffer);
01721      break;
01722      }
01723      g_free (buffer);
01724      }
01725      else
01726      {
01727          g_free (buffer);
01728          break;
01729      }
01730      }
01731
01732  // Freeing and closing
01733  g_free (name);
01734  g_free (directory);
01735  gtk_widget_destroy (GTK_WIDGET (dlg));
01736  #if DEBUG
01737  fprintf (stderr, "window_open: end\n");
01738  #endif
01739  }
01740
01745  void
01746  window_new ()
01747  {
01748      unsigned int i;
01749      char *buffer, *buffer2, buffer3[64];
01750      char *label_algorithm[NALGORITHMS] = {
01751          "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
01752      };
01753      char *tip_algorithm[NALGORITHMS] = {
01754          gettext ("Monte-Carlo brute force algorithm"),
01755          gettext ("Sweep brute force algorithm"),
01756          gettext ("Genetic algorithm")
01757      };
01758      char *label_direction[N DIRECTIONS] = {
01759          gettext ("_Coordinates descent"), gettext ("_Random")
01760      };
01761      char *tip_direction[N DIRECTIONS] = {
01762          gettext ("Coordinates direction estimate method"),

```



```

01763     gettext ("Random direction estimate method")
01764 };
01765 char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
01766 char *tip_norm[NNORMS] = {
01767     gettext ("Euclidean error norm (L2)"),
01768     gettext ("Maximum error norm (L)"),
01769     gettext ("P error norm (Lp)"),
01770     gettext ("Taxicab error norm (L1)")
01771 };
01772
01773 #if DEBUG
01774 fprintf (stderr, "window_new: start\n");
01775 #endif
01776
01777 // Creating the window
01778 window->window = main_window
01779     = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
01780
01781 // Finish when closing the window
01782 g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
01783
01784 // Setting the window title
01785 gtk_window_set_title (window->window, "MPCOTool");
01786
01787 // Creating the open button
01788 window->button_open = (GtkToolButton *) gtk_tool_button_new
01789     (gtk_image_new_from_icon_name ("document-open",
01790         GTK_ICON_SIZE_LARGE_TOOLBAR),
01791     gettext ("Open"));
01792 g_signal_connect (window->button_open, "clicked", window_open, NULL);
01793
01794 // Creating the save button
01795 window->button_save = (GtkToolButton *) gtk_tool_button_new
01796     (gtk_image_new_from_icon_name ("document-save",
01797         GTK_ICON_SIZE_LARGE_TOOLBAR),
01798     gettext ("Save"));
01799 g_signal_connect (window->button_save, "clicked", (void (*)(
01800 window_save,
01801     NULL));
01802
01803 // Creating the run button
01804 window->button_run = (GtkToolButton *) gtk_tool_button_new
01805     (gtk_image_new_from_icon_name ("system-run",
01806         GTK_ICON_SIZE_LARGE_TOOLBAR),
01807     gettext ("Run"));
01808 g_signal_connect (window->button_run, "clicked", window_run, NULL);
01809
01810 // Creating the options button
01811 window->button_options = (GtkToolButton *) gtk_tool_button_new
01812     (gtk_image_new_from_icon_name ("preferences-system",
01813         GTK_ICON_SIZE_LARGE_TOOLBAR),
01814     gettext ("Options"));
01815 g_signal_connect (window->button_options, "clicked", options_new, NULL);
01816
01817 // Creating the help button
01818 window->button_help = (GtkToolButton *) gtk_tool_button_new
01819     (gtk_image_new_from_icon_name ("help-browser",
01820         GTK_ICON_SIZE_LARGE_TOOLBAR),
01821     gettext ("Help"));
01822 g_signal_connect (window->button_help, "clicked", window_help, NULL);
01823
01824 // Creating the about button
01825 window->button_about = (GtkToolButton *) gtk_tool_button_new
01826     (gtk_image_new_from_icon_name ("help-about",
01827         GTK_ICON_SIZE_LARGE_TOOLBAR),
01828     gettext ("About"));
01829 g_signal_connect (window->button_about, "clicked", window_about, NULL);
01830
01831 // Creating the exit button
01832 window->button_exit = (GtkToolButton *) gtk_tool_button_new
01833     (gtk_image_new_from_icon_name ("application-exit",
01834         GTK_ICON_SIZE_LARGE_TOOLBAR),
01835     gettext ("Exit"));
01836 g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
01837
01838 // Creating the buttons bar
01839 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
01840 gtk_toolbar_insert
01841     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
01842 gtk_toolbar_insert
01843     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
01844 gtk_toolbar_insert
01845     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
01846 gtk_toolbar_insert
01847     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
01848 gtk_toolbar_insert
01849     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);

```

```

01849 gtk_toolbar_insert
01850     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
01851 gtk_toolbar_insert
01852     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
01853 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
01854
01855 // Creating the simulator program label and entry
01856 window->label_simulator
01857     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
01858 window->button_simulator = (GtkFileChooserButton *)
01859     gtk_file_chooser_button_new (gettext ("Simulator program"),
01860     GTK_FILE_CHOOSER_ACTION_OPEN);
01861 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
01862     gettext ("Simulator program executable file"));
01863 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
01864
01865 // Creating the evaluator program label and entry
01866 window->check_evaluator = (GtkCheckButton *)
01867     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
01868 g_signal_connect (window->check_evaluator, "toggled",
01869     window_update, NULL);
01869 window->button_evaluator = (GtkFileChooserButton *)
01870     gtk_file_chooser_button_new (gettext ("Evaluator program"),
01871     GTK_FILE_CHOOSER_ACTION_OPEN);
01872 gtk_widget_set_tooltip_text
01873     (GTK_WIDGET (window->button_evaluator),
01874     gettext ("Optional evaluator program executable file"));
01875
01876 // Creating the results files labels and entries
01877 window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
01878 window->entry_result = (GtkEntry *) gtk_entry_new ();
01879 gtk_widget_set_tooltip_text
01880     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
01881 window->label_variables
01882     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
01883 window->entry_variables = (GtkEntry *) gtk_entry_new ();
01884 gtk_widget_set_tooltip_text
01885     (GTK_WIDGET (window->entry_variables),
01886     gettext ("All simulated results file"));
01887
01888 // Creating the files grid and attaching widgets
01889 window->grid_files = (GtkGrid *) gtk_grid_new ();
01890 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01891     label_simulator),
01892     0, 0, 1, 1);
01893 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01894     button_simulator),
01895     1, 0, 1, 1);
01896 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01897     check_evaluator),
01898     0, 1, 1, 1);
01899 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01900     button_evaluator),
01901     1, 1, 1, 1);
01902 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01903     label_result),
01904     0, 2, 1, 1);
01905 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01906     entry_result),
01907     1, 2, 1, 1);
01908 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01909     label_variables),
01910     0, 3, 1, 1);
01911 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
01912     entry_variables),
01913     1, 3, 1, 1);
01914
01915 // Creating the algorithm properties
01916 window->label_simulations = (GtkLabel *) gtk_label_new
01917     (gettext ("Simulations number"));
01918 window->spin_simulations
01919     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
01920 gtk_widget_set_tooltip_text
01921     (GTK_WIDGET (window->spin_simulations),
01922     gettext ("Number of simulations to perform for each iteration"));
01923 gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_simulations), TRUE);
01924 window->label_iterations = (GtkLabel *)
01925     gtk_label_new (gettext ("Iterations number"));
01926 window->spin_iterations
01927     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01928 gtk_widget_set_tooltip_text
01929     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
01930 g_signal_connect
01931     (window->spin_iterations, "value-changed", window_update, NULL);
01932 gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_iterations), TRUE);
01933 window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
01934 window->spin_tolerance

```

```

01927     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01928     gtk_widget_set_tooltip_text
01929     (GTK_WIDGET (window->spin_tolerance),
01930      gettext ("Tolerance to set the variable interval on the next iteration"));
01931     window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
01932     window->spin_bests
01933     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01934     gtk_widget_set_tooltip_text
01935     (GTK_WIDGET (window->spin_bests),
01936      gettext ("Number of best simulations used to set the variable interval "
01937               "on the next iteration"));
01938     window->label_population
01939     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
01940     window->spin_population
01941     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
01942     gtk_widget_set_tooltip_text
01943     (GTK_WIDGET (window->spin_population),
01944      gettext ("Number of population for the genetic algorithm"));
01945     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
01946     window->label_generations
01947     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
01948     window->spin_generations
01949     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
01950     gtk_widget_set_tooltip_text
01951     (GTK_WIDGET (window->spin_generations),
01952      gettext ("Number of generations for the genetic algorithm"));
01953     window->label_mutation
01954     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
01955     window->spin_mutation
01956     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01957     gtk_widget_set_tooltip_text
01958     (GTK_WIDGET (window->spin_mutation),
01959      gettext ("Ratio of mutation for the genetic algorithm"));
01960     window->label_reproduction
01961     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
01962     window->spin_reproduction
01963     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01964     gtk_widget_set_tooltip_text
01965     (GTK_WIDGET (window->spin_reproduction),
01966      gettext ("Ratio of reproduction for the genetic algorithm"));
01967     window->label_adaptation
01968     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
01969     window->spin_adaptation
01970     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
01971     gtk_widget_set_tooltip_text
01972     (GTK_WIDGET (window->spin_adaptation),
01973      gettext ("Ratio of adaptation for the genetic algorithm"));
01974     window->label_threshold = (GtkLabel *) gtk_label_new (gettext ("Threshold"));
01975     window->spin_threshold = (GtkSpinButton *) gtk_spin_button_new_with_range
01976     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
01977     gtk_widget_set_tooltip_text
01978     (GTK_WIDGET (window->spin_threshold),
01979      gettext ("Threshold in the objective function to finish the simulations"));
01980     window->scrolled_threshold
01981     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
01982     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
01983                       GTK_WIDGET (window->spin_threshold));
01984     // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
01985     // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
01986     //                          GTK_ALIGN_FILL);
01987
01988     // Creating the direction search method properties
01989     window->check_direction = (GtkCheckButton *)
01990     gtk_check_button_new_with_mnemonic (gettext ("_Direction search method"));
01991     g_signal_connect (window->check_direction, "clicked",
01992                      window_update, NULL);
01993     window->grid_direction = (GtkGrid *) gtk_grid_new ();
01994     window->button_direction[0] = (GtkRadioButton *)
01995     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
01996     gtk_grid_attach (window->grid_direction,
01997                     GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
01998     g_signal_connect (window->button_direction[0], "clicked",
01999                      window_update, NULL);
02000     for (i = 0; ++i < NDIRECTIONS;)
02001     {
02002         window->button_direction[i] = (GtkRadioButton *)
02003         gtk_radio_button_new_with_mnemonic
02004         (gtk_radio_button_get_group (window->button_direction[0]),
02005          label_direction[i]);
02006         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02007                                     tip_direction[i]);
02008         gtk_grid_attach (window->grid_direction,
02009                         GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02010         g_signal_connect (window->button_direction[i], "clicked",
02011                          window_update, NULL);
02012     }

```

```

02012 window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
02013 window->spin_steps = (GtkSpinButton *)
02014     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02015 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02016 window->label_estimates
02017     = (GtkLabel *) gtk_label_new (gettext ("Direction estimates number"));
02018 window->spin_estimates = (GtkSpinButton *)
02019     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02020 window->label_relaxation
02021     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
02022 window->spin_relaxation = (GtkSpinButton *)
02023     gtk_spin_button_new_with_range (0., 2., 0.001);
02024 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
label_steps),
02025     0, NDIRECTIONS, 1, 1);
02026 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
spin_steps),
02027     1, NDIRECTIONS, 1, 1);
02028 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
label_estimates),
02029     0, NDIRECTIONS + 1, 1, 1);
02030 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
spin_estimates),
02031     1, NDIRECTIONS + 1, 1, 1);
02032 gtk_grid_attach (window->grid_direction,
02033     GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2, 1,
02034     1);
02035 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
spin_relaxation),
02036     1, NDIRECTIONS + 2, 1, 1);
02037
02038 // Creating the array of algorithms
02039 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02040 window->button_algorithm[0] = (GtkRadioButton *)
02041     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02042 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02043     tip_algorithm[0]);
02044 gtk_grid_attach (window->grid_algorithm,
02045     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02046 g_signal_connect (window->button_algorithm[0], "clicked",
02047     window_set_algorithm, NULL);
02048 for (i = 0; ++i < NALGORITHMS;)
02049     {
02050         window->button_algorithm[i] = (GtkRadioButton *)
02051             gtk_radio_button_new_with_mnemonic
02052                 (gtk_radio_button_get_group (window->button_algorithm[0]),
02053                 label_algorithm[i]);
02054         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02055             tip_algorithm[i]);
02056         gtk_grid_attach (window->grid_algorithm,
02057             GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02058         g_signal_connect (window->button_algorithm[i], "clicked",
02059             window_set_algorithm, NULL);
02060     }
02061 gtk_grid_attach (window->grid_algorithm,
02062     GTK_WIDGET (window->label_simulations), 0,
02063     NALGORITHMS, 1, 1);
02064 gtk_grid_attach (window->grid_algorithm,
02065     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02066 gtk_grid_attach (window->grid_algorithm,
02067     GTK_WIDGET (window->label_iterations), 0,
02068     NALGORITHMS + 1, 1, 1);
02069 gtk_grid_attach (window->grid_algorithm,
02070     GTK_WIDGET (window->spin_iterations), 1,
02071     NALGORITHMS + 1, 1, 1);
02072 gtk_grid_attach (window->grid_algorithm,
02073     GTK_WIDGET (window->label_tolerance), 0,
02074     NALGORITHMS + 2, 1, 1);
02075 gtk_grid_attach (window->grid_algorithm,
02076     GTK_WIDGET (window->spin_tolerance), 1,
02077     NALGORITHMS + 2, 1, 1);
02078 gtk_grid_attach (window->grid_algorithm,
02079     GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
02080 gtk_grid_attach (window->grid_algorithm,
02081     GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
02082 gtk_grid_attach (window->grid_algorithm,
02083     GTK_WIDGET (window->label_population), 0,
02084     NALGORITHMS + 4, 1, 1);
02085 gtk_grid_attach (window->grid_algorithm,
02086     GTK_WIDGET (window->spin_population), 1,
02087     NALGORITHMS + 4, 1, 1);
02088 gtk_grid_attach (window->grid_algorithm,
02089     GTK_WIDGET (window->label_generations), 0,
02090     NALGORITHMS + 5, 1, 1);
02091 gtk_grid_attach (window->grid_algorithm,
02092     GTK_WIDGET (window->spin_generations), 1,
02093     NALGORITHMS + 5, 1, 1);

```

```

02094 gtk_grid_attach (window->grid_algorithm,
02095                 GTK_WIDGET (window->label_mutation), 0,
02096                 NALGORITHMS + 6, 1, 1);
02097 gtk_grid_attach (window->grid_algorithm,
02098                 GTK_WIDGET (window->spin_mutation), 1,
02099                 NALGORITHMS + 6, 1, 1);
02100 gtk_grid_attach (window->grid_algorithm,
02101                 GTK_WIDGET (window->label_reproduction), 0,
02102                 NALGORITHMS + 7, 1, 1);
02103 gtk_grid_attach (window->grid_algorithm,
02104                 GTK_WIDGET (window->spin_reproduction), 1,
02105                 NALGORITHMS + 7, 1, 1);
02106 gtk_grid_attach (window->grid_algorithm,
02107                 GTK_WIDGET (window->label_adaptation), 0,
02108                 NALGORITHMS + 8, 1, 1);
02109 gtk_grid_attach (window->grid_algorithm,
02110                 GTK_WIDGET (window->spin_adaptation), 1,
02111                 NALGORITHMS + 8, 1, 1);
02112 gtk_grid_attach (window->grid_algorithm,
02113                 GTK_WIDGET (window->check_direction), 0,
02114                 NALGORITHMS + 9, 2, 1);
02115 gtk_grid_attach (window->grid_algorithm,
02116                 GTK_WIDGET (window->grid_direction), 0,
02117                 NALGORITHMS + 10, 2, 1);
02118 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_threshold),
02119                 0, NALGORITHMS + 11, 1, 1);
02120 gtk_grid_attach (window->grid_algorithm,
02121                 GTK_WIDGET (window->scrolled_threshold), 1,
02122                 NALGORITHMS + 11, 1, 1);
02123 window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
02124 gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02125                 GTK_WIDGET (window->grid_algorithm));
02126
02127 // Creating the variable widgets
02128 window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02129 gtk_widget_set_tooltip_text
02130     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
02131 window->id_variable = g_signal_connect
02132     (window->combo_variable, "changed", window_set_variable, NULL);
02133 window->button_add_variable
02134     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02135                 GTK_ICON_SIZE_BUTTON);
02136 g_signal_connect
02137     (window->button_add_variable, "clicked",
02138     window_add_variable, NULL);
02139 gtk_widget_set_tooltip_text
02140     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
02141 window->button_remove_variable
02142     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02143                 GTK_ICON_SIZE_BUTTON);
02144 g_signal_connect
02145     (window->button_remove_variable, "clicked",
02146     window_remove_variable, NULL);
02147 gtk_widget_set_tooltip_text
02148     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
02149 window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
02150 window->entry_variable = (GtkEntry *) gtk_entry_new ();
02151 gtk_widget_set_tooltip_text
02152     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
02153 gtk_widget_set_hexexpand (GTK_WIDGET (window->entry_variable), TRUE);
02154 window->id_variable_label = g_signal_connect
02155     (window->entry_variable, "changed", window_label_variable, NULL);
02156 window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
02157 window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02158     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02159 gtk_widget_set_tooltip_text
02160     (GTK_WIDGET (window->spin_min),
02161     gettext ("Minimum initial value of the variable"));
02162 window->scrolled_min
02163     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02164 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02165                 GTK_WIDGET (window->spin_min));
02166 g_signal_connect (window->spin_min, "value-changed",
02167                 window_rangemin_variable, NULL);
02168 window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
02169 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02170     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02171 gtk_widget_set_tooltip_text
02172     (GTK_WIDGET (window->spin_max),
02173     gettext ("Maximum initial value of the variable"));
02174 window->scrolled_max
02175     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02176 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02177                 GTK_WIDGET (window->spin_max));
02178 g_signal_connect (window->spin_max, "value-changed",
02179                 window_rangemax_variable, NULL);

```

```

02178 window->check_minabs = (GtkCheckButton *)
02179     gtk_check_button_new_with_mnemonic (gettext ("Absolute minimum"));
02180 g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02181 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02182     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02183 gtk_widget_set_tooltip_text
02184     (GTK_WIDGET (window->spin_minabs),
02185      gettext ("Minimum allowed value of the variable"));
02186 window->scrolled_minabs
02187     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02188 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02189     GTK_WIDGET (window->spin_minabs));
02190 g_signal_connect (window->spin_minabs, "value-changed",
02191     window_rangeminabs_variable, NULL);
02192 window->check_maxabs = (GtkCheckButton *)
02193     gtk_check_button_new_with_mnemonic (gettext ("Absolute maximum"));
02194 g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02195 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02196     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02197 gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->spin_maxabs),
02199      gettext ("Maximum allowed value of the variable"));
02200 window->scrolled_maxabs
02201     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02202 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02203     GTK_WIDGET (window->spin_maxabs));
02204 g_signal_connect (window->spin_maxabs, "value-changed",
02205     window_rangemaxabs_variable, NULL);
02206 window->label_precision
02207     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
02208 window->spin_precision = (GtkSpinButton *)
02209     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02210 gtk_widget_set_tooltip_text
02211     (GTK_WIDGET (window->spin_precision),
02212      gettext ("Number of precision floating point digits\n"
02213              "0 is for integer numbers"));
02214 g_signal_connect (window->spin_precision, "value-changed",
02215     window_precision_variable, NULL);
02216 window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
02217 window->spin_sweeps
02218     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02219 gtk_widget_set_tooltip_text
02220     (GTK_WIDGET (window->spin_sweeps),
02221      gettext ("Number of steps sweeping the variable"));
02222 g_signal_connect
02223     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
02224 window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
02225 window->spin_bits
02226     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02227 gtk_widget_set_tooltip_text
02228     (GTK_WIDGET (window->spin_bits),
02229      gettext ("Number of bits to encode the variable"));
02230 g_signal_connect
02231     (window->spin_bits, "value-changed", window_update_variable, NULL);
02232 window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
02233 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02234     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02235 gtk_widget_set_tooltip_text
02236     (GTK_WIDGET (window->spin_step),
02237      gettext ("Initial step size for the direction search method"));
02238 window->scrolled_step
02239     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02240 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02241     GTK_WIDGET (window->spin_step));
02242 g_signal_connect
02243     (window->spin_step, "value-changed", window_step_variable, NULL);
02244 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02245 gtk_grid_attach (window->grid_variable,
02246     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02247 gtk_grid_attach (window->grid_variable,
02248     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02249 gtk_grid_attach (window->grid_variable,
02250     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02251 gtk_grid_attach (window->grid_variable,
02252     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02253 gtk_grid_attach (window->grid_variable,
02254     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02255 gtk_grid_attach (window->grid_variable,
02256     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02257 gtk_grid_attach (window->grid_variable,
02258     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02259 gtk_grid_attach (window->grid_variable,
02260     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02261 gtk_grid_attach (window->grid_variable,
02262     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02263 gtk_grid_attach (window->grid_variable,
02264     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);

```



```

02265 gtk_grid_attach (window->grid_variable,
02266                 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02267 gtk_grid_attach (window->grid_variable,
02268                 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02269 gtk_grid_attach (window->grid_variable,
02270                 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02271 gtk_grid_attach (window->grid_variable,
02272                 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02273 gtk_grid_attach (window->grid_variable,
02274                 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02275 gtk_grid_attach (window->grid_variable,
02276                 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02277 gtk_grid_attach (window->grid_variable,
02278                 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02279 gtk_grid_attach (window->grid_variable,
02280                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02281 gtk_grid_attach (window->grid_variable,
02282                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02283 gtk_grid_attach (window->grid_variable,
02284                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02285 gtk_grid_attach (window->grid_variable,
02286                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02287 window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
02288 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02289                  GTK_WIDGET (window->grid_variable));
02290
02291 // Creating the experiment widgets
02292 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02293 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02294                             gettext ("Experiment selector"));
02295 window->id_experiment = g_signal_connect
02296 (window->combo_experiment, "changed", window_set_experiment, NULL)
;
02297 window->button_add_experiment
02298 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02299                                               GTK_ICON_SIZE_BUTTON);
02300 g_signal_connect
02301 (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02302 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02303                             gettext ("Add experiment"));
02304 window->button_remove_experiment
02305 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02306                                               GTK_ICON_SIZE_BUTTON);
02307 g_signal_connect (window->button_remove_experiment, "clicked",
02308                  window_remove_experiment, NULL);
02309 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02310                             gettext ("Remove experiment"));
02311 window->label_experiment
02312 = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
02313 window->button_experiment = (GtkFileChooserButton *)
02314   gtk_file_chooser_button_new (gettext ("Experimental data file"),
02315                               GTK_FILE_CHOOSER_ACTION_OPEN);
02316 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02317                             gettext ("Experimental data file"));
02318 window->id_experiment_name
02319 = g_signal_connect (window->button_experiment, "selection-changed",
02320                    window_name_experiment, NULL);
02321 gtk_widget_set_expand (GTK_WIDGET (window->button_experiment), TRUE);
02322 window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
02323 window->spin_weight
02324 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02325 gtk_widget_set_tooltip_text
02326 (GTK_WIDGET (window->spin_weight),
02327  gettext ("Weight factor to build the objective function"));
02328 g_signal_connect
02329 (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
02330 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02331 gtk_grid_attach (window->grid_experiment,
02332                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02333 gtk_grid_attach (window->grid_experiment,
02334                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02335 gtk_grid_attach (window->grid_experiment,
02336                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02337 gtk_grid_attach (window->grid_experiment,
02338                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02339 gtk_grid_attach (window->grid_experiment,
02340                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02341 gtk_grid_attach (window->grid_experiment,
02342                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02343 gtk_grid_attach (window->grid_experiment,
02344                 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02345 for (i = 0; i < MAX_NINPUTS; ++i)
02346 {
02347     snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02348     window->check_template[i] = (GtkCheckButton *)

```

```

02349     gtk_check_button_new_with_label (buffer3);
02350     window->id_template[i]
02351     = g_signal_connect (window->check_template[i], "toggled",
02352                        window_inputs_experiment, NULL);
02353     gtk_grid_attach (window->grid_experiment,
02354                    GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02355     window->button_template[i] = (GtkFileChooserButton *)
02356     gtk_file_chooser_button_new (gettext ("Input template"),
02357                                GTK_FILE_CHOOSER_ACTION_OPEN);
02358     gtk_widget_set_tooltip_text
02359     (GTK_WIDGET (window->button_template[i]),
02360      gettext ("Experimental input template file"));
02361     window->id_input[i]
02362     = g_signal_connect_swapped (window->button_template[i],
02363                                "selection-changed",
02364                                (void (*)(void *)) window_template_experiment,
02365                                (void *) (size_t) i);
02366     gtk_grid_attach (window->grid_experiment,
02367                    GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02368 }
02369 window->frame_experiment
02370 = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02371 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02372                  GTK_WIDGET (window->grid_experiment));
02373
02374 // Creating the error norm widgets
02375 window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
02376 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02377 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02378                  GTK_WIDGET (window->grid_norm));
02379 window->button_norm[0] = (GtkRadioButton *)
02380     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02381 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02382                             tip_norm[0]);
02383 gtk_grid_attach (window->grid_norm,
02384                 GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02385 g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02386 for (i = 0; ++i < NNORMS;)
02387 {
02388     window->button_norm[i] = (GtkRadioButton *)
02389     gtk_radio_button_new_with_mnemonic
02390     (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02391     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02392                                tip_norm[i]);
02393     gtk_grid_attach (window->grid_norm,
02394                    GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02395     g_signal_connect (window->button_norm[i], "clicked",
02396                      window_update, NULL);
02397 }
02398 window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02399 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02400 window->spin_p = (GtkSpinButton *)
02401     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02402 gtk_widget_set_tooltip_text
02403 (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02404 window->scrolled_p
02405 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02406 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02407                  GTK_WIDGET (window->spin_p));
02408 gtk_widget_set_hexpan (GTK_WIDGET (window->scrolled_p), TRUE);
02409 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02410 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02411                1, 2, 1, 2);
02412
02413 // Creating the grid and attaching the widgets to the grid
02414 window->grid = (GtkGrid *) gtk_grid_new ();
02415 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02416 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02417 gtk_grid_attach (window->grid,
02418                GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02419 gtk_grid_attach (window->grid,
02420                GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02421 gtk_grid_attach (window->grid,
02422                GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02423 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02424 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
02425 grid));
02426
02427 // Setting the window logo
02428 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02429 gtk_window_set_icon (window->window, window->logo);
02430
02431 // Showing the window
02432 gtk_widget_show_all (GTK_WIDGET (window->window));
02433
02434 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02435 #if GTK_MINOR_VERSION >= 16

```



```

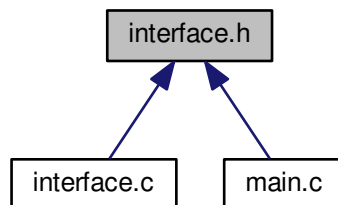
02434  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02435  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02436  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02437  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02438  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02439  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02440  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02441  #endif
02442
02443  // Reading initial example
02444  input_new ();
02445  buffer2 = g_get_current_dir ();
02446  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02447  g_free (buffer2);
02448  window_read (buffer);
02449  g_free (buffer);
02450
02451  #if DEBUG
02452  fprintf (stderr, "window_new: start\n");
02453  #endif
02454  }

```

## 5.13 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](#) ()  
*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.*

### 5.13.1 Detailed Description

Header file to define the graphical interface functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.13.2 Function Documentation

#### 5.13.2.1 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 352 of file [utils.c](#).

```

00353 {
00354     unsigned int i;
00355     for (i = 0; i < n; ++i)
00356         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00357             break;
00358     return i;
00359 }
```

## 5.13.2.2 void input\_save ( char \* filename )

Function to save the input file.

## Parameters

<i>filename</i>	<a href="#">Input</a> file name.
-----------------	----------------------------------

Definition at line 204 of file [interface.c](#).

```

00205 {
00206     unsigned int i, j;
00207     char *buffer;
00208     xmlDoc *doc;
00209     xmlNode *node, *child;
00210     GFile *file, *file2;
00211
00212     #if DEBUG
00213         fprintf (stderr, "input_save: start\n");
00214     #endif
00215
00216     // Getting the input file directory
00217     input->name = g_path_get_basename (filename);
00218     input->directory = g_path_get_dirname (filename);
00219     file = g_file_new_for_path (input->directory);
00220
00221     // Opening the input file
00222     doc = xmlNewDoc ((const xmlChar *) "1.0");
00223
00224     // Setting root XML node
00225     node = xmlNewDocNode (doc, 0, XML_OPTIMIZE, 0);
00226     xmlDocSetRootElement (doc, node);
00227
00228     // Adding properties to the root XML node
00229     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
00230         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
00231     if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
00232         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
00233     file2 = g_file_new_for_path (input->simulator);
00234     buffer = g_file_get_relative_path (file, file2);
00235     g_object_unref (file2);
00236     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
00237     g_free (buffer);
00238     if (input->evaluator)
00239     {
00240         file2 = g_file_new_for_path (input->evaluator);
00241         buffer = g_file_get_relative_path (file, file2);
00242         g_object_unref (file2);
00243         if (xmlStrlen ((xmlChar *) buffer))
00244             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
00245         g_free (buffer);
00246     }
00247     if (input->seed != DEFAULT_RANDOM_SEED)
00248         xml_node_set_uint (node, XML_SEED, input->seed);
00249
00250     // Setting the algorithm
00251     buffer = (char *) g_malloc (64);
```

```

00252     switch (input->algorithm)
00253     {
00254         case ALGORITHM_MONTE_CARLO:
00255             xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
00256             snprintf (buffer, 64, "%u", input->nsimulations);
00257             xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
00258             snprintf (buffer, 64, "%u", input->niterations);
00259             xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00260             snprintf (buffer, 64, "%.3lg", input->tolerance);
00261             xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00262             snprintf (buffer, 64, "%u", input->nbest);
00263             xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00264             input_save_direction (node);
00265             break;
00266         case ALGORITHM_SWEEP:
00267             xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
00268             snprintf (buffer, 64, "%u", input->niterations);
00269             xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
00270             snprintf (buffer, 64, "%.3lg", input->tolerance);
00271             xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
00272             snprintf (buffer, 64, "%u", input->nbest);
00273             xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
00274             input_save_direction (node);
00275             break;
00276         default:
00277             xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
00278             snprintf (buffer, 64, "%u", input->nsimulations);
00279             xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
00280             snprintf (buffer, 64, "%u", input->niterations);
00281             xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
00282             snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00283             xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
00284             snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00285             xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
00286             snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00287             xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
00288             break;
00289     }
00290     g_free (buffer);
00291     if (input->threshold != 0.)
00292         xml_node_set_float (node, XML_THRESHOLD, input->
00293                             threshold);
00294     // Setting the experimental data
00295     for (i = 0; i < input->nexperiments; ++i)
00296     {
00297         child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
00298         xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i].
00299                     name);
00300         if (input->experiment[i].weight != 1.)
00301             xml_node_set_float (child, XML_WEIGHT, input->
00302                                 experiment[i].weight);
00303         for (j = 0; j < input->experiment->ninputs; ++j)
00304             xmlSetProp (child, template[j],
00305                         (xmlChar *) input->experiment[i].template[j]);
00306     }
00307     // Setting the variables data
00308     for (i = 0; i < input->nvariables; ++i)
00309     {
00310         child = xmlNewChild (node, 0, XML_VARIABLE, 0);
00311         xmlSetProp (child, XML_NAME, (xmlChar *) input->variable[i].
00312                     name);
00313         xml_node_set_float (child, XML_MINIMUM, input->
00314                             variable[i].rangemin);
00315         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00316             xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
00317                                 input->variable[i].rangeminabs);
00318         xml_node_set_float (child, XML_MAXIMUM, input->
00319                             variable[i].rangemax);
00320         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00321             xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
00322                                 input->variable[i].rangemaxabs);
00323         if (input->variable[i].precision != DEFAULT_PRECISION)
00324             xml_node_set_uint (child, XML_PRECISION, input->
00325                                 variable[i].precision);
00326         if (input->algorithm == ALGORITHM_SWEEP)
00327             xml_node_set_uint (child, XML_NSWEEPS, input->
00328                                 variable[i].nsweeps);
00329         else if (input->algorithm == ALGORITHM_GENETIC)
00330             xml_node_set_uint (child, XML_NBITS, input->
00331                                 variable[i].nbits);
00332         if (input->nsteps)
00333             xml_node_set_float (child, XML_STEP, input->
00334                                 variable[i].step);
00335     }
00336 }

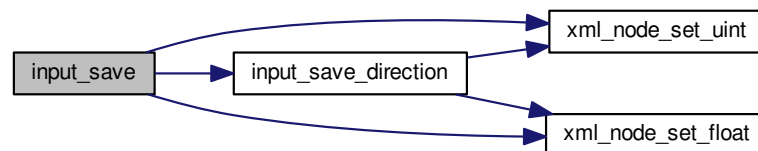
```

```

00329 // Saving the error norm
00330 switch (input->norm)
00331 {
00332     case ERROR_NORM_MAXIMUM:
00333         xmlSetProp (node, XML_NORM, XML_MAXIMUM);
00334         break;
00335     case ERROR_NORM_P:
00336         xmlSetProp (node, XML_NORM, XML_P);
00337         xml_node_set_float (node, XML_P, input->p);
00338         break;
00339     case ERROR_NORM_TAXICAB:
00340         xmlSetProp (node, XML_NORM, XML_TAXICAB);
00341 }
00342
00343 // Saving the XML file
00344 xmlSaveFormatFile (filename, doc, 1);
00345
00346 // Freeing memory
00347 xmlFreeDoc (doc);
00348
00349 #if DEBUG
00350 fprintf (stderr, "input_save: end\n");
00351 #endif
00352 }

```

Here is the call graph for this function:



### 5.13.2.3 unsigned int window\_get\_algorithm ( )

Function to get the stochastic algorithm number.

#### Returns

Stochastic algorithm number.

Definition at line 461 of file [interface.c](#).

```

00462 {
00463     unsigned int i;
00464     #if DEBUG
00465         fprintf (stderr, "window_get_algorithm: start\n");
00466     #endif
00467     i = gtk_array_get_active (window->button_algorithm,
00468                             NAALGORITHMS);
00469     #if DEBUG
00470         fprintf (stderr, "window_get_algorithm: %u\n", i);
00471     #endif
00472     return i;
00473 }

```

Here is the call graph for this function:



#### 5.13.2.4 unsigned int window\_get\_direction ( )

Function to get the direction search method number.

##### Returns

Direction search method number.

Definition at line 481 of file [interface.c](#).

```
00482 {  
00483     unsigned int i;  
00484     #if DEBUG  
00485     fprintf (stderr, "window_get_direction: start\n");  
00486     #endif  
00487     i = gtk_array_get_active (window->button_direction,  
00488                             NDIRECTIONS);  
00488     #if DEBUG  
00489     fprintf (stderr, "window_get_direction: %u\n", i);  
00490     fprintf (stderr, "window_get_direction: end\n");  
00491     #endif  
00492     return i;  
00493 }
```

Here is the call graph for this function:



#### 5.13.2.5 unsigned int window\_get\_norm ( )

Function to get the norm method number.

##### Returns

Norm method number.

Definition at line 501 of file [interface.c](#).

```

00502 {
00503     unsigned int i;
00504     #if DEBUG
00505         fprintf (stderr, "window_get_norm: start\n");
00506     #endif
00507     i = gtk_array_get_active (window->button_norm,
NNORMS);
00508     #if DEBUG
00509         fprintf (stderr, "window_get_norm: %u\n", i);
00510         fprintf (stderr, "window_get_norm: end\n");
00511     #endif
00512     return i;
00513 }

```

Here is the call graph for this function:



#### 5.13.2.6 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 1560 of file [interface.c](#).

```

01561 {
01562     unsigned int i;
01563     char *buffer;
01564     #if DEBUG
01565         fprintf (stderr, "window_read: start\n");
01566     #endif
01567
01568     // Reading new input file
01569     input_free ();
01570     if (!input_open (filename))
01571         return 0;
01572
01573     // Setting GTK+ widgets data
01574     gtk_entry_set_text (window->entry_result, input->result);
01575     gtk_entry_set_text (window->entry_variables, input->
variables);
01576     buffer = g_build_filename (input->directory, input->simulator, NULL);
01577     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01578     g_free (buffer);
01579     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01580
01581     if (input->evaluator)
01582     {
01583         buffer = g_build_filename (input->directory, input->evaluator, NULL);
01584         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01585         g_free (buffer);
01586     }
01587     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->

```

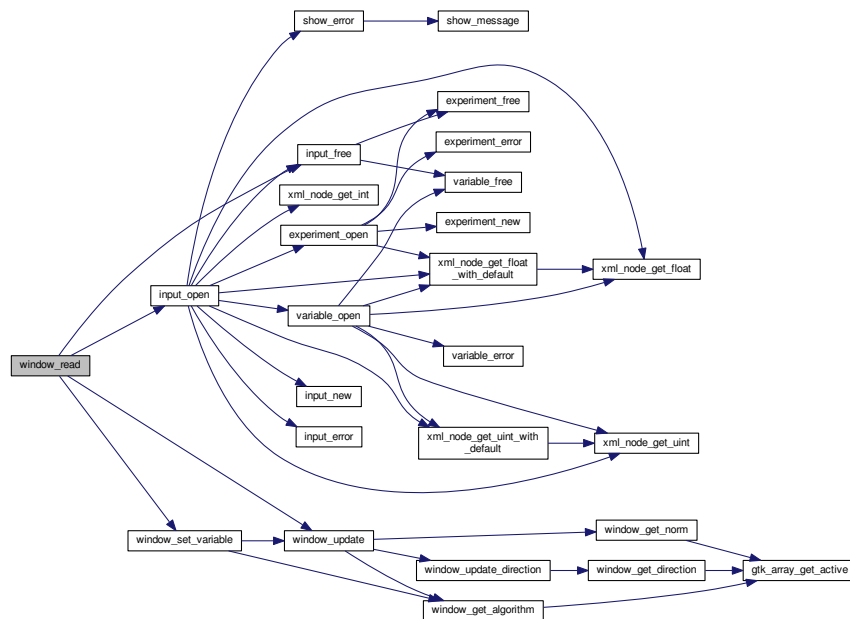


```

    algorithm)), TRUE);
01591     switch (input->algorithm)
01592     {
01593         case ALGORITHM_MONTE_CARLO:
01594             gtk_spin_button_set_value (window->spin_simulations,
01595                                         (gdouble) input->nsimulations);
01596         case ALGORITHM_SWEEP:
01597             gtk_spin_button_set_value (window->spin_iterations,
01598                                         (gdouble) input->niterations);
01599             gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01600             gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01601             gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_direction),
                                         input->nsteps);
01602             if (input->nsteps)
01603             {
01604                 gtk_toggle_button_set_active
01605                     (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01606                 gtk_spin_button_set_value (window->spin_steps,
01607                                             (gdouble) input->nsteps);
01608                 gtk_spin_button_set_value (window->spin_relaxation,
01609                                             (gdouble) input->relaxation);
01610                 switch (input->direction)
01611                 {
01612                     case DIRECTION_METHOD_RANDOM:
01613                         gtk_spin_button_set_value (window->spin_estimates,
01614                                                     (gdouble) input->nestimates);
01615                     }
01616                 break;
01617             }
01618             default:
01619                 gtk_spin_button_set_value (window->spin_population,
01620                                             (gdouble) input->nsimulations);
01621                 gtk_spin_button_set_value (window->spin_generations,
01622                                             (gdouble) input->niterations);
01623                 gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01624                 gtk_spin_button_set_value (window->spin_reproduction,
01625                                             input->reproduction_ratio);
01626                 gtk_spin_button_set_value (window->spin_adaptation,
01627                                             input->adaptation_ratio);
01628             }
01629             gtk_toggle_button_set_active
01630                 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01631             gtk_spin_button_set_value (window->spin_p, input->p);
01632             gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01633             g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01634             g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
01635             gtk_combo_box_text_remove_all (window->combo_experiment);
01636             for (i = 0; i < input->nexperiments; ++i)
01637                 gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i].name);
01638             g_signal_handler_unblock
01639                 (window->button_experiment, window->
id_experiment_name);
01640             g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01641             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01642             g_signal_handler_block (window->combo_variable, window->
id_variable);
01643             g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01644             gtk_combo_box_text_remove_all (window->combo_variable);
01645             for (i = 0; i < input->nvariables; ++i)
01646                 gtk_combo_box_text_append_text (window->combo_variable,
input->variable[i].name);
01647             g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01648             g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01649             gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01650             window_set_variable ();
01651             window_update ();
01652         }
01653     }
01654     #if DEBUG
01655     fprintf (stderr, "window_read: end\n");
01656     #endif
01657     return 1;
01658 }

```

Here is the call graph for this function:



### 5.13.2.7 int window\_save ( )

Function to save the input file.

#### Returns

1 on OK, 0 on Cancel.

Definition at line 554 of file [interface.c](#).

```

00555 {
00556     GtkFileChooserDialog *dlg;
00557     GtkFileFilter *filter;
00558     char *buffer;
00559
00560     #if DEBUG
00561         fprintf (stderr, "window_save: start\n");
00562     #endif
00563
00564     // Opening the saving dialog
00565     dlg = (GtkFileChooserDialog *)
00566         gtk_file_chooser_dialog_new (gettext ("Save file"),
00567                                     window->window,
00568                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00569                                     gettext ("_Cancel"),
00570                                     GTK_RESPONSE_CANCEL,
00571                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00572     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00573     buffer = g_build_filename (input->directory, input->name, NULL);
00574     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00575     g_free (buffer);
00576
00577     // Adding XML filter
00578     filter = (GtkFileFilter *) gtk_file_filter_new ();
00579     gtk_file_filter_set_name (filter, "XML");
00580     gtk_file_filter_add_pattern (filter, "*.xml");
00581     gtk_file_filter_add_pattern (filter, "*.XML");
00582     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
00583
00584     // If OK response then saving
00585     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)

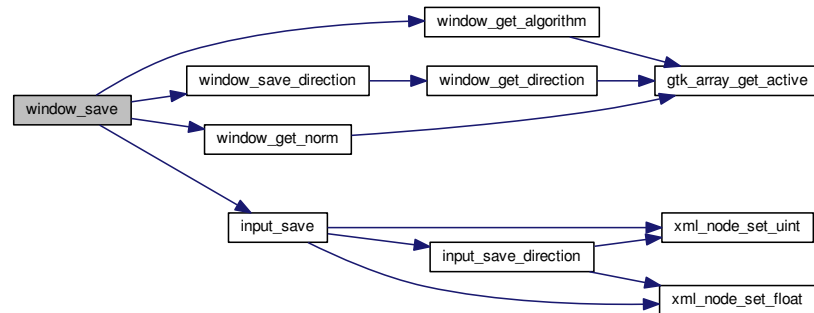
```

```

00586     {
00587
00588         // Adding properties to the root XML node
00589         input->simulator = gtk_file_chooser_get_filename
00590             (GTK_FILE_CHOOSER (window->button_simulator));
00591         if (gtk_toggle_button_get_active
00592             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00593             input->evaluator = gtk_file_chooser_get_filename
00594                 (GTK_FILE_CHOOSER (window->button_evaluator));
00595         else
00596             input->evaluator = NULL;
00597         input->result
00598             = (char *) xmlStrdup ((const xmlChar *)
00599                 gtk_entry_get_text (window->entry_result));
00600         input->variables
00601             = (char *) xmlStrdup ((const xmlChar *)
00602                 gtk_entry_get_text (window->entry_variables));
00603
00604         // Setting the algorithm
00605         switch (window_get_algorithm ())
00606         {
00607             case ALGORITHM_MONTE_CARLO:
00608                 input->algorithm = ALGORITHM_MONTE_CARLO;
00609                 input->nsimulations
00610                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
00611                 input->niterations
00612                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00613                 input->tolerance = gtk_spin_button_get_value (window->
00614                     spin_tolerance);
00615                 input->nbest = gtk_spin_button_get_value_as_int (window->
00616                     spin_bests);
00617                 window_save_direction ();
00618                 break;
00619             case ALGORITHM_SWEEP:
00620                 input->algorithm = ALGORITHM_SWEEP;
00621                 input->niterations
00622                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
00623                 input->tolerance = gtk_spin_button_get_value (window->
00624                     spin_tolerance);
00625                 input->nbest = gtk_spin_button_get_value_as_int (window->
00626                     spin_bests);
00627                 window_save_direction ();
00628                 break;
00629             default:
00630                 input->algorithm = ALGORITHM_GENETIC;
00631                 input->nsimulations
00632                     = gtk_spin_button_get_value_as_int (window->spin_population);
00633                 input->niterations
00634                     = gtk_spin_button_get_value_as_int (window->spin_generations);
00635                 input->mutation_ratio
00636                     = gtk_spin_button_get_value (window->spin_mutation);
00637                 input->reproduction_ratio
00638                     = gtk_spin_button_get_value (window->spin_reproduction);
00639                 input->adaptation_ratio
00640                     = gtk_spin_button_get_value (window->spin_adaptation);
00641                 break;
00642         }
00643         input->norm = window_get_norm ();
00644         input->p = gtk_spin_button_get_value (window->spin_p);
00645         input->threshold = gtk_spin_button_get_value (window->
00646             spin_threshold);
00647
00648         // Saving the XML file
00649         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00650         input_save (buffer);
00651
00652         // Closing and freeing memory
00653         g_free (buffer);
00654         gtk_widget_destroy (GTK_WIDGET (dlg));
00655 #if DEBUG
00656         fprintf (stderr, "window_save: end\n");
00657 #endif
00658         return 1;
00659     }
00660
00661     // Closing and freeing memory
00662     gtk_widget_destroy (GTK_WIDGET (dlg));
00663 #if DEBUG
00664     fprintf (stderr, "window_save: end\n");
00665 #endif
00666     return 0;
00667 }

```

Here is the call graph for this function:



### 5.13.2.8 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 1210 of file [interface.c](#).

```

01211 {
01212     unsigned int i, j;
01213     char *buffer;
01214     GFile *file1, *file2;
01215     #if DEBUG
01216         fprintf (stderr, "window_template_experiment: start\n");
01217     #endif
01218     i = (size_t) data;
01219     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01220     file1
01221         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01222     file2 = g_file_new_for_path (input->directory);
01223     buffer = g_file_get_relative_path (file2, file1);
01224     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01225     g_free (buffer);
01226     g_object_unref (file2);
01227     g_object_unref (file1);
01228     #if DEBUG
01229         fprintf (stderr, "window_template_experiment: end\n");
01230     #endif
01231 }

```

## 5.14 interface.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the

```

```

00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INTERFACE__H
00033 #define INTERFACE__H 1
00034
00035 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00036
00037 typedef struct
00038 {
00039     GtkWidget *dialog;
00040     GtkWidget *grid;
00041     GtkWidget *label_seed;
00042     GtkWidget *spin_seed;
00043     GtkWidget *label_threads;
00044     GtkWidget *spin_threads;
00045     GtkWidget *label_direction;
00046     GtkWidget *spin_direction;
00047 } Options;
00048
00049 typedef struct
00050 {
00051     GtkWidget *dialog;
00052     GtkWidget *label;
00053     GtkWidget *spinner;
00054     GtkWidget *grid;
00055 } Running;
00056
00057 typedef struct
00058 {
00059     GtkWidget *window;
00060     GtkWidget *grid;
00061     GtkWidget *bar_buttons;
00062     GtkWidget *button_open;
00063     GtkWidget *button_save;
00064     GtkWidget *button_run;
00065     GtkWidget *button_options;
00066     GtkWidget *button_help;
00067     GtkWidget *button_about;
00068     GtkWidget *button_exit;
00069     GtkWidget *grid_files;
00070     GtkWidget *label_simulator;
00071     GtkWidget *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *button_evaluator;
00074     GtkWidget *label_result;
00075     GtkWidget *entry_result;
00076     GtkWidget *label_variables;
00077     GtkWidget *entry_variables;
00078     GtkWidget *frame_norm;
00079     GtkWidget *grid_norm;
00080     GtkWidget *button_norm[NORMS];
00081     GtkWidget *label_p;
00082     GtkWidget *spin_p;
00083     GtkWidget *scrolled_p;
00084     GtkWidget *frame_algorithm;
00085     GtkWidget *grid_algorithm;
00086     GtkWidget *button_algorithm[NALGORITHMS];
00087     GtkWidget *label_simulations;
00088     GtkWidget *spin_simulations;
00089     GtkWidget *label_iterations;
00090     GtkWidget *spin_iterations;
00091     GtkWidget *label_tolerance;
00092     GtkWidget *spin_tolerance;
00093     GtkWidget *label_bests;
00094     GtkWidget *spin_bests;
00095     GtkWidget *label_population;
00096     GtkWidget *spin_population;
00097     GtkWidget *label_generations;
00098     GtkWidget *spin_generations;
00099     GtkWidget *label_mutation;
00100     GtkWidget *spin_mutation;
00101     GtkWidget *label_reproduction;
00102     GtkWidget *spin_reproduction;
00103     GtkWidget *label_adaptation;
00104     GtkWidget *spin_adaptation;

```

```

00137   GtkWidget *check_direction;
00139   GtkWidget *grid_direction;
00141   GtkWidget *button_direction[N DIRECTIONS];
00143   GtkWidget *label_steps;
00144   GtkWidget *spin_steps;
00145   GtkWidget *label_estimates;
00146   GtkWidget *spin_estimates;
00148   GtkWidget *label_relaxation;
00150   GtkWidget *spin_relaxation;
00152   GtkWidget *label_thresold;
00153   GtkWidget *spin_thresold;
00154   GtkWidget *scrolled_thresold;
00156   GtkWidget *frame_variable;
00157   GtkWidget *grid_variable;
00158   GtkWidget *comboBoxText *combo_variable;
00160   GtkWidget *button_add_variable;
00161   GtkWidget *button_remove_variable;
00162   GtkWidget *label_variable;
00163   GtkWidget *entry_variable;
00164   GtkWidget *label_min;
00165   GtkWidget *spin_min;
00166   GtkWidget *scrolled_min;
00167   GtkWidget *label_max;
00168   GtkWidget *spin_max;
00169   GtkWidget *scrolled_max;
00170   GtkWidget *check_minabs;
00171   GtkWidget *spin_minabs;
00172   GtkWidget *scrolled_minabs;
00173   GtkWidget *check_maxabs;
00174   GtkWidget *spin_maxabs;
00175   GtkWidget *scrolled_maxabs;
00176   GtkWidget *label_precision;
00177   GtkWidget *spin_precision;
00178   GtkWidget *label_sweeps;
00179   GtkWidget *spin_sweeps;
00180   GtkWidget *label_bits;
00181   GtkWidget *spin_bits;
00182   GtkWidget *label_step;
00183   GtkWidget *spin_step;
00184   GtkWidget *scrolled_step;
00185   GtkWidget *frame_experiment;
00186   GtkWidget *grid_experiment;
00187   GtkWidget *comboBoxText *combo_experiment;
00188   GtkWidget *button_add_experiment;
00189   GtkWidget *button_remove_experiment;
00190   GtkWidget *label_experiment;
00191   GtkWidget *FileChooserButton *button_experiment;
00193   GtkWidget *label_weight;
00194   GtkWidget *spin_weight;
00195   GtkWidget *check_template[MAX_NINPUTS];
00197   GtkWidget *FileChooserButton *button_template[MAX_NINPUTS];
00199   GdkPixbuf *logo;
00200   Experiment *experiment;
00201   Variable *variable;
00202   char *application_directory;
00203   gulong id_experiment;
00204   gulong id_experiment_name;
00205   gulong id_variable;
00206   gulong id_variable_label;
00207   gulong id_template[MAX_NINPUTS];
00209   gulong id_input[MAX_NINPUTS];
00211   unsigned int nexperiments;
00212   unsigned int nvariables;
00213 } Window;
00214
00215 // Global variables
00216 extern const char *logo[];
00217 extern Options options[1];
00218 extern Running running[1];
00219 extern Window window[1];
00220
00221 // Public functions
00222 unsigned int gtk_array_get_active (GtkWidget * array[], unsigned int n);
00223 void input_save (char *filename);
00224 void options_new ();
00225 void running_new ();
00226 unsigned int window_get_algorithm ();
00227 unsigned int window_get_direction ();
00228 unsigned int window_get_norm ();
00229 void window_save_direction ();
00230 int window_save ();
00231 void window_run ();
00232 void window_help ();
00233 void window_update_direction ();
00234 void window_update ();
00235 void window_set_algorithm ();
00236 void window_set_experiment ();

```

```

00237 void window_remove_experiment ();
00238 void window_add_experiment ();
00239 void window_name_experiment ();
00240 void window_weight_experiment ();
00241 void window_inputs_experiment ();
00242 void window_template_experiment (void *data);
00243 void window_set_variable ();
00244 void window_remove_variable ();
00245 void window_add_variable ();
00246 void window_label_variable ();
00247 void window_precision_variable ();
00248 void window_rangemin_variable ();
00249 void window_rangemax_variable ();
00250 void window_rangeminabs_variable ();
00251 void window_rangemaxabs_variable ();
00252 void window_update_variable ();
00253 int window_read (char *filename);
00254 void window_open ();
00255 void window_new ();
00256
00257 #endif

```

## 5.15 main.c File Reference

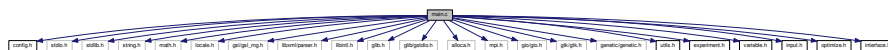
Main source file.

```

#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 <glib/gstdio.h>
#include <alloca.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 main.c:



## Macros

- `#define _GNU_SOURCE`
- `#define DEBUG 0`

*Macro to debug.*

## Functions

- int [main](#) (int argn, char \*\*argc)  
*Main function.*

### 5.15.1 Detailed Description

Main source file.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.15.2 Function Documentation

#### 5.15.2.1 int main ( int *argn*, char \*\* *argc* )

Main function.

##### Parameters

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

##### Returns

0 on success, >0 on error.

Definition at line [84](#) of file [main.c](#).

```
00085 {
00086     #if HAVE_GTK
00087     char *buffer;
00088     #endif
00089
00090     // Starting pseudo-random numbers generator
00091     #if DEBUG
00092     fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00093     #endif
00094     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00095
00096     // Allowing spaces in the XML data file
00097     #if DEBUG
00098     fprintf (stderr, "main: allowing spaces in the XML data file\n");
00099     #endif
00100     xmlKeepBlanksDefault (0);
00101
00102     // Starting MPI
00103     #if HAVE_MPI
00104     #if DEBUG
00105     fprintf (stderr, "main: starting MPI\n");
00106     #endif
00107     MPI_Init (&argn, &argc);
00108     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00109     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00110     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00111     #else
00112     ntasks = 1;
00113     #endif
00114
00115     // Resetting result and variables file names
```



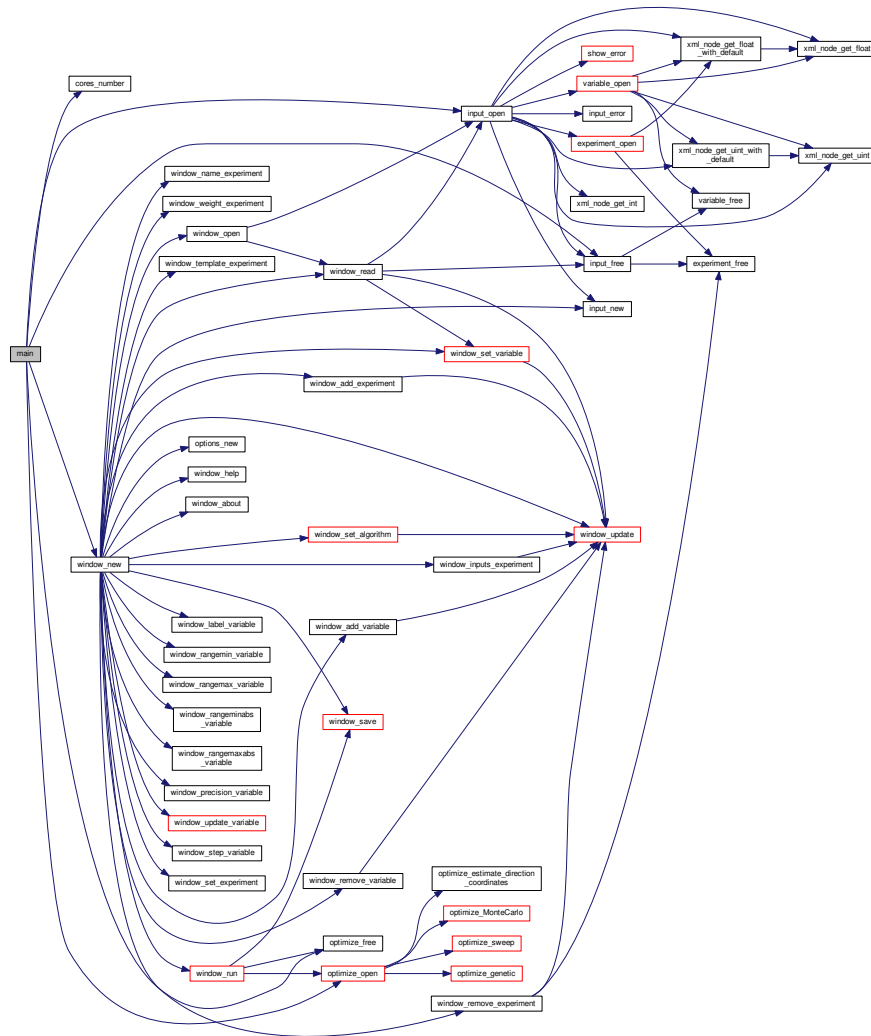
```

00116 #if DEBUG
00117     fprintf (stderr, "main: resetting result and variables file names\n");
00118 #endif
00119     input->result = input->variables = NULL;
00120
00121 #if HAVE_GTK
00122
00123     // Getting threads number and pseudo-random numbers generator seed
00124     nthreads_direction = nthreads = cores_number ();
00125     optimize->seed = DEFAULT_RANDOM_SEED;
00126
00127     // Setting local language and international floating point numbers notation
00128     setlocale (LC_ALL, "");
00129     setlocale (LC_NUMERIC, "C");
00130     window->application_directory = g_get_current_dir ();
00131     buffer = g_build_filename (window->application_directory,
00132                               LOCALE_DIR, NULL);
00133     bindtextdomain (PROGRAM_INTERFACE, buffer);
00134     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00135     textdomain (PROGRAM_INTERFACE);
00136
00137     // Initing GTK+
00138     gtk_disable_setlocale ();
00139     gtk_init (&argn, &argc);
00140
00141     // Opening the main window
00142     window_new ();
00143     gtk_main ();
00144
00145     // Freeing memory
00146     input_free ();
00147     g_free (buffer);
00148     gtk_widget_destroy (GTK_WIDGET (window->window));
00149     g_free (window->application_directory);
00150 #else
00151
00152     // Checking syntax
00153     if (argn < 2)
00154     {
00155         printf ("The syntax is:\n"
00156                "    ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00157                "[variables_file]\n");
00158         return 1;
00159     }
00160
00161     // Getting threads number and pseudo-random numbers generator seed
00162 #if DEBUG
00163     fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00164             "generator seed\n");
00165 #endif
00166     nthreads_direction = nthreads = cores_number ();
00167     optimize->seed = DEFAULT_RANDOM_SEED;
00168     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00169     {
00170         nthreads_direction = nthreads = atoi (argc[2]);
00171         if (!nthreads)
00172         {
00173             printf ("Bad threads number\n");
00174             return 2;
00175         }
00176         argc += 2;
00177         argn -= 2;
00178         if (argn > 2 && !strcmp (argc[1], "-seed"))
00179         {
00180             optimize->seed = atoi (argc[2]);
00181             argc += 2;
00182             argn -= 2;
00183         }
00184     }
00185     else if (argn > 2 && !strcmp (argc[1], "-seed"))
00186     {
00187         optimize->seed = atoi (argc[2]);
00188         argc += 2;
00189         argn -= 2;
00190         if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00191         {
00192             nthreads_direction = nthreads = atoi (argc[2]);
00193             if (!nthreads)
00194             {
00195                 printf ("Bad threads number\n");
00196                 return 2;
00197             }
00198             argc += 2;
00199             argn -= 2;
00200         }
00201     }

```

```
00202     printf ("nthreads=%u\n", nthreads);
00203     printf ("seed=%lu\n", optimize->seed);
00204
00205     // Checking arguments
00206     #if DEBUG
00207     fprintf (stderr, "main: checking arguments\n");
00208     #endif
00209     if (argn > 4 || argn < 2)
00210     {
00211         printf ("The syntax is:\n"
00212                "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00213                "[variables_file]\n");
00214         return 1;
00215     }
00216     if (argn > 2)
00217         input->result = argc[2];
00218     if (argn == 4)
00219         input->variables = argc[3];
00220
00221     // Making optimization
00222     #if DEBUG
00223     fprintf (stderr, "main: making optimization\n");
00224     #endif
00225     if (input_open (argc[1]))
00226         optimize_open ();
00227
00228     // Freeing memory
00229     #if DEBUG
00230     fprintf (stderr, "main: freeing memory and closing\n");
00231     #endif
00232     optimize_free ();
00233
00234     #endif
00235
00236     // Closing MPI
00237     #if HAVE_MPI
00238     MPI_Finalize ();
00239     #endif
00240
00241     // Freeing memory
00242     gsl_rng_free (optimize->rng);
00243
00244     // Closing
00245     return 0;
00246 }
```

Here is the call graph for this function:



## 5.16 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 2012-2016, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR

```

```

00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.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 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #elif !defined (BSD)
00047 #include <alloca.h>
00048 #endif
00049 #if HAVE_MPI
00050 #include <mpi.h>
00051 #endif
00052 #if HAVE_GTK
00053 #include <gio/gio.h>
00054 #include <gtk/gtk.h>
00055 #endif
00056 #include "genetic/genetic.h"
00057 #include "utils.h"
00058 #include "experiment.h"
00059 #include "variable.h"
00060 #include "input.h"
00061 #include "optimize.h"
00062 #if HAVE_GTK
00063 #include "interface.h"
00064 #endif
00065
00066 #define DEBUG 0
00067
00068 int
00069 main (int argn, char **argc)
00070 {
00071     #if HAVE_GTK
00072         char *buffer;
00073     #endif
00074
00075     // Starting pseudo-random numbers generator
00076     #if DEBUG
00077         fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00078     #endif
00079     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00080
00081     // Allowing spaces in the XML data file
00082     #if DEBUG
00083         fprintf (stderr, "main: allowing spaces in the XML data file\n");
00084     #endif
00085     xmlKeepBlanksDefault (0);
00086
00087     // Starting MPI
00088     #if HAVE_MPI
00089     #if DEBUG
00090         fprintf (stderr, "main: starting MPI\n");
00091     #endif
00092     MPI_Init (&argn, &argc);
00093     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00094     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00095     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00096     #else
00097         ntasks = 1;
00098     #endif
00099
00100     // Resetting result and variables file names
00101     #if DEBUG
00102         fprintf (stderr, "main: resetting result and variables file names\n");
00103     #endif
00104     input->result = input->variables = NULL;
00105
00106     #if HAVE_GTK
00107
00108     // Getting threads number and pseudo-random numbers generator seed
00109     nthreads_direction = nthreads = cores_number ();
00110     optimize->seed = DEFAULT_RANDOM_SEED;
00111

```

```

00127 // Setting local language and international floating point numbers notation
00128 setlocale (LC_ALL, "");
00129 setlocale (LC_NUMERIC, "C");
00130 window->application_directory = g_get_current_dir ();
00131 buffer = g_build_filename (window->application_directory,
    LOCALE_DIR, NULL);
00132 bindtextdomain (PROGRAM_INTERFACE, buffer);
00133 bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00134 textdomain (PROGRAM_INTERFACE);
00135
00136 // Initing GTK+
00137 gtk_disable_setlocale ();
00138 gtk_init (&argn, &argc);
00139
00140 // Opening the main window
00141 window_new ();
00142 gtk_main ();
00143
00144 // Freeing memory
00145 input_free ();
00146 g_free (buffer);
00147 gtk_widget_destroy (GTK_WIDGET (window->window));
00148 g_free (window->application_directory);
00149
00150 #else
00151
00152 // Checking syntax
00153 if (argn < 2)
00154 {
00155     printf ("The syntax is:\n"
00156            " ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00157            "[variables_file]\n");
00158     return 1;
00159 }
00160
00161 // Getting threads number and pseudo-random numbers generator seed
00162 #if DEBUG
00163 fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00164         "generator seed\n");
00165 #endif
00166 nthreads_direction = nthreads = cores_number ();
00167 optimize->seed = DEFAULT_RANDOM_SEED;
00168 if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00169 {
00170     nthreads_direction = nthreads = atoi (argc[2]);
00171     if (!nthreads)
00172     {
00173         printf ("Bad threads number\n");
00174         return 2;
00175     }
00176     argc += 2;
00177     argn -= 2;
00178     if (argn > 2 && !strcmp (argc[1], "-seed"))
00179     {
00180         optimize->seed = atoi (argc[2]);
00181         argc += 2;
00182         argn -= 2;
00183     }
00184 }
00185 else if (argn > 2 && !strcmp (argc[1], "-seed"))
00186 {
00187     optimize->seed = atoi (argc[2]);
00188     argc += 2;
00189     argn -= 2;
00190     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00191     {
00192         nthreads_direction = nthreads = atoi (argc[2]);
00193         if (!nthreads)
00194         {
00195             printf ("Bad threads number\n");
00196             return 2;
00197         }
00198         argc += 2;
00199         argn -= 2;
00200     }
00201 }
00202 printf ("nthreads=%u\n", nthreads);
00203 printf ("seed=%lu\n", optimize->seed);
00204
00205 // Checking arguments
00206 #if DEBUG
00207 fprintf (stderr, "main: checking arguments\n");
00208 #endif
00209 if (argn > 4 || argn < 2)
00210 {
00211     printf ("The syntax is:\n"
00212            " ./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "

```

```

00213         "[variables_file]\n");
00214     return 1;
00215 }
00216 if (argn > 2)
00217     input->result = argc[2];
00218 if (argn == 4)
00219     input->variables = argc[3];
00220
00221 // Making optimization
00222 #if DEBUG
00223     fprintf (stderr, "main: making optimization\n");
00224 #endif
00225 if (input_open (argc[1]))
00226     optimize_open ();
00227
00228 // Freeing memory
00229 #if DEBUG
00230     fprintf (stderr, "main: freeing memory and closing\n");
00231 #endif
00232     optimize_free ();
00233
00234 #endif
00235
00236 // Closing MPI
00237 #if HAVE_MPI
00238     MPI_Finalize ();
00239 #endif
00240
00241 // Freeing memory
00242     gsl_rng_free (optimize->rng);
00243
00244 // Closing
00245     return 0;
00246 }

```

## 5.17 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 <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.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 _GNU_SOURCE`

- `#define DEBUG 0`  
*Macro to debug.*
- `#define RM "rm"`  
*Macro to define the shell remove command.*

## Functions

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

- Function to optimize with a direction search method.*

  - double [optimize\\_genetic\\_objective](#) (Entity \*entity)

*Function to calculate the objective function of an entity.*
- void [optimize\\_genetic](#) ()

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

*Function to save the best results on iterative methods.*
- void [optimize\\_merge\\_old](#) ()

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

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

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

*Function to iterate the algorithm.*
- void [optimize\\_free](#) ()

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

*Function to open and perform a optimization.*

## Variables

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

### 5.17.1 Detailed Description

Source file to define the optimization functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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



## 5.17.2 Function Documentation

### 5.17.2.1 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 461 of file [optimize.c](#).

```

00462 {
00463     unsigned int i, j;
00464     double e;
00465     #if DEBUG
00466         fprintf (stderr, "optimize_best: start\n");
00467         fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00468                 optimize->nsaveds, optimize->nbest);
00469     #endif
00470     if (optimize->nsaveds < optimize->nbest
00471         || value < optimize->error_best[optimize->nsaveds - 1])
00472     {
00473         if (optimize->nsaveds < optimize->nbest)
00474             ++optimize->nsaveds;
00475         optimize->error_best[optimize->nsaveds - 1] = value;
00476         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00477         for (i = optimize->nsaveds; --i;)
00478         {
00479             if (optimize->error_best[i] < optimize->
00480                 error_best[i - 1])
00481             {
00482                 j = optimize->simulation_best[i];
00483                 e = optimize->error_best[i];
00484                 optimize->simulation_best[i] = optimize->
00485                     simulation_best[i - 1];
00486                 optimize->error_best[i] = optimize->
00487                     error_best[i - 1];
00488                 optimize->simulation_best[i - 1] = j;
00489                 optimize->error_best[i - 1] = e;
00490             }
00491             else
00492                 break;
00493         }
00494     }
00495     #if DEBUG
00496         fprintf (stderr, "optimize_best: end\n");
00497     #endif
00498 }
```

### 5.17.2.2 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 786 of file [optimize.c](#).

```

00787 {
00788     #if DEBUG
00789         fprintf (stderr, "optimize_best_direction: start\n");
00790         fprintf (stderr,
00791                 "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00792                 simulation, value, optimize->error_best[0]);
00793     #endif
00794     if (value < optimize->error_best[0])
00795     {
00796         optimize->error_best[0] = value;
00797         optimize->simulation_best[0] = simulation;
00798     }
00799     #if DEBUG
00800         fprintf (stderr,
00801                 "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00802                 simulation, value);
00803     #endif
00804     #if DEBUG
00805         fprintf (stderr, "optimize_best_direction: end\n");
00806     #endif
00807 }
```

### 5.17.2.3 void optimize\_direction\_sequential ( unsigned int *simulation* )

Function to estimate the direction search sequentially.

## Parameters

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

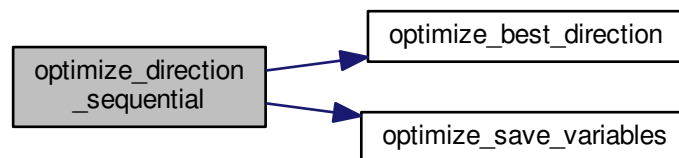
Definition at line 816 of file [optimize.c](#).

```

00817 {
00818     unsigned int i, j;
00819     double e;
00820     #if DEBUG
00821     fprintf (stderr, "optimize_direction_sequential: start\n");
00822     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00823             "nend_direction=%u\n",
00824             optimize->nstart_direction, optimize->
nend_direction);
00825     #endif
00826     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00827     {
00828         j = simulation + i;
00829         e = optimize_norm (j);
00830         optimize_best_direction (j, e);
00831         optimize_save_variables (j, e);
00832         if (e < optimize->threshold)
00833         {
00834             optimize->stop = 1;
00835             break;
00836         }
00837     #if DEBUG
00838     fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00839     #endif
00840     }
00841     #if DEBUG
00842     fprintf (stderr, "optimize_direction_sequential: end\n");
00843     #endif
00844 }

```

Here is the call graph for this function:



#### 5.17.2.4 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 854 of file [optimize.c](#).

```

00855 {
00856     unsigned int i, thread;

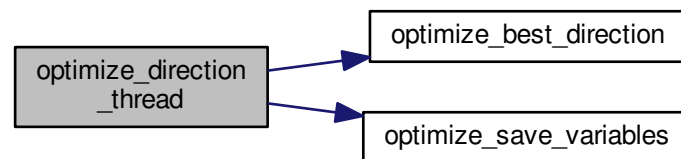
```

```

00857     double e;
00858     #if DEBUG
00859     fprintf (stderr, "optimize_direction_thread: start\n");
00860     #endif
00861     thread = data->thread;
00862     #if DEBUG
00863     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00864             thread,
00865             optimize->thread_direction[thread],
00866             optimize->thread_direction[thread + 1]);
00867     #endif
00868     for (i = optimize->thread_direction[thread];
00869          i < optimize->thread_direction[thread + 1]; ++i)
00870     {
00871         e = optimize_norm (i);
00872         g_mutex_lock (mutex);
00873         optimize_best_direction (i, e);
00874         optimize_save_variables (i, e);
00875         if (e < optimize->thresold)
00876             optimize->stop = 1;
00877         g_mutex_unlock (mutex);
00878         if (optimize->stop)
00879             break;
00880     #if DEBUG
00881     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00882     #endif
00883     }
00884     #if DEBUG
00885     fprintf (stderr, "optimize_direction_thread: end\n");
00886     #endif
00887     g_thread_exit (NULL);
00888     return NULL;
00889 }

```

Here is the call graph for this function:



#### 5.17.2.5 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 928 of file [optimize.c](#).

```

00930 {
00931     double x;
00932     #if DEBUG
00933     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00934     #endif
00935     x = optimize->direction[variable];
00936     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00937     {
00938         if (estimate & 1)
00939             x += optimize->step[variable];
00940         else

```

```

00941         x -= optimize->step[variable];
00942     }
00943     #if DEBUG
00944     fprintf (stderr,
00945             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00946             variable, x);
00947     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00948     #endif
00949     return x;
00950 }

```

#### 5.17.2.6 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 901 of file [optimize.c](#).

```

00903 {
00904     double x;
00905     #if DEBUG
00906     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907     #endif
00908     x = optimize->direction[variable]
00909         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00910         step[variable];
00911     #if DEBUG
00912     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00913             variable, x);
00914     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00915     #endif
00916     return x;
00917 }

```

#### 5.17.2.7 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 1095 of file [optimize.c](#).

```

01096 {
01097     unsigned int j;
01098     double objective;
01099     char buffer[64];
01100     #if DEBUG
01101     fprintf (stderr, "optimize_genetic_objective: start\n");
01102     #endif
01103     for (j = 0; j < optimize->nvariables; ++j)
01104     {
01105         optimize->value[entity->id * optimize->nvariables + j]
01106             = genetic_get_variable (entity, optimize->genetic_variable + j);
01107     }
01108     objective = optimize_norm (entity->id);
01109     g_mutex_lock (mutex);
01110     for (j = 0; j < optimize->nvariables; ++j)
01111     {
01112         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01113         fprintf (optimize->file_variables, buffer,
01114                 genetic_get_variable (entity, optimize->genetic_variable + j));
01115     }
01116 }

```

```

01115     }
01116     fprintf (optimize->file_variables, "%.14le\n", objective);
01117     g_mutex_unlock (mutex);
01118 #if DEBUG
01119     fprintf (stderr, "optimize_genetic_objective: end\n");
01120 #endif
01121     return objective;
01122 }

```

#### 5.17.2.8 void optimize\_input ( unsigned int *simulation*, char \* *input*, GMappedFile \* *template* )

Function to write the simulation input file.

##### Parameters

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

Definition at line 102 of file [optimize.c](#).

```

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

```

```

00161     g_free (buffer2);
00162     g_regex_unref (regex);
00163 }
00164
00165 // Saving input file
00166 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00167 g_free (buffer3);
00168 fclose (file);
00169
00170 optimize_input_end:
00171 #if DEBUG
00172     fprintf (stderr, "optimize_input: end\n");
00173 #endif
00174     return;
00175 }

```

#### 5.17.2.9 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 584 of file [optimize.c](#).

```

00586 {
00587     unsigned int i, j, k, s[optimize->nbest];
00588     double e[optimize->nbest];
00589 #if DEBUG
00590     fprintf (stderr, "optimize_merge: start\n");
00591 #endif
00592     i = j = k = 0;
00593     do
00594     {
00595         if (i == optimize->nsaveds)
00596         {
00597             s[k] = simulation_best[j];
00598             e[k] = error_best[j];
00599             ++j;
00600             ++k;
00601             if (j == nsaveds)
00602                 break;
00603         }
00604         else if (j == nsaveds)
00605         {
00606             s[k] = optimize->simulation_best[i];
00607             e[k] = optimize->error_best[i];
00608             ++i;
00609             ++k;
00610             if (i == optimize->nsaveds)
00611                 break;
00612         }
00613         else if (optimize->error_best[i] > error_best[j])
00614         {
00615             s[k] = simulation_best[j];
00616             e[k] = error_best[j];
00617             ++j;
00618             ++k;
00619         }
00620         else
00621         {
00622             s[k] = optimize->simulation_best[i];
00623             e[k] = optimize->error_best[i];
00624             ++i;
00625             ++k;
00626         }
00627     }
00628     while (k < optimize->nbest);
00629     optimize->nsaveds = k;
00630     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00631     memcpy (optimize->error_best, e, k * sizeof (double));
00632 #if DEBUG
00633     fprintf (stderr, "optimize_merge: end\n");
00634 #endif
00635 }

```



5.17.2.10 double optimize\_norm\_euclidian ( unsigned int *simulation* )

Function to calculate the Euclidian error norm.

## Parameters

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

## Returns

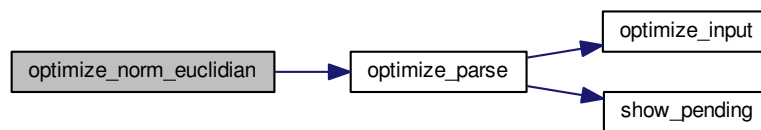
Euclidian error norm.

Definition at line 294 of file [optimize.c](#).

```

00295 {
00296     double e, ei;
00297     unsigned int i;
00298     #if DEBUG
00299     fprintf (stderr, "optimize_norm_euclidian: start\n");
00300     #endif
00301     e = 0.;
00302     for (i = 0; i < optimize->nexperiments; ++i)
00303     {
00304         ei = optimize_parse (simulation, i);
00305         e += ei * ei;
00306     }
00307     e = sqrt (e);
00308     #if DEBUG
00309     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00310     fprintf (stderr, "optimize_norm_euclidian: end\n");
00311     #endif
00312     return e;
00313 }
```

Here is the call graph for this function:



#### 5.17.2.11 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 323 of file [optimize.c](#).

```

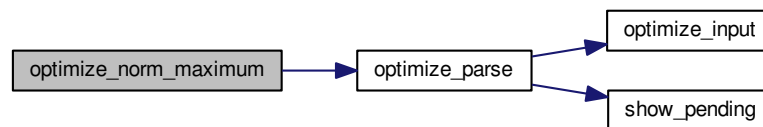
00324 {
00325     double e, ei;
00326     unsigned int i;
00327     #if DEBUG
00328     fprintf (stderr, "optimize_norm_maximum: start\n");
00329     #endif
00330     e = 0.;
00331     for (i = 0; i < optimize->nexperiments; ++i)
00332     {
```

```

00333     ei = fabs (optimize_parse (simulation, i));
00334     e = fmax (e, ei);
00335 }
00336 #if DEBUG
00337 fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00338 fprintf (stderr, "optimize_norm_maximum: end\n");
00339 #endif
00340 return e;
00341 }

```

Here is the call graph for this function:



#### 5.17.2.12 double optimize\_norm\_p ( unsigned int *simulation* )

Function to calculate the P error norm.

##### Parameters

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

##### Returns

P error norm.

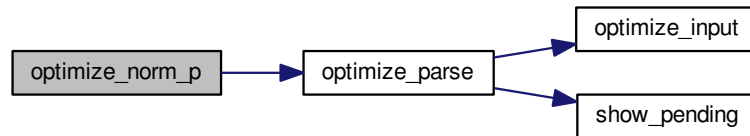
Definition at line 351 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:



#### 5.17.2.13 double optimize\_norm\_taxicab ( unsigned int *simulation* )

Function to calculate the taxicab error norm.

##### Parameters

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

##### Returns

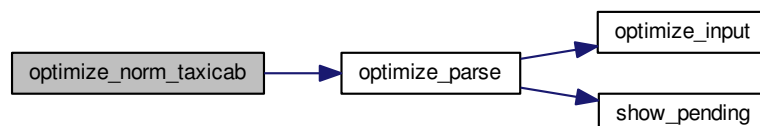
Taxicab error norm.

Definition at line 380 of file [optimize.c](#).

```

00381 {
00382     double e;
00383     unsigned int i;
00384     #if DEBUG
00385     fprintf (stderr, "optimize_norm_taxicab: start\n");
00386     #endif
00387     e = 0.;
00388     for (i = 0; i < optimize->nexperiments; ++i)
00389         e += fabs (optimize_parse (simulation, i));
00390     #if DEBUG
00391     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00392     fprintf (stderr, "optimize_norm_taxicab: end\n");
00393     #endif
00394     return e;
00395 }
  
```

Here is the call graph for this function:



#### 5.17.2.14 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>	<a href="#">Experiment</a> number.

## Returns

Objective function value.

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

```

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

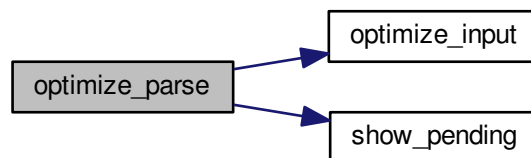
```

```

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

```

Here is the call graph for this function:



#### 5.17.2.15 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 433 of file [optimize.c](#).

```

00434 {
00435     unsigned int i;
00436     char buffer[64];
00437     #if DEBUG
00438     fprintf (stderr, "optimize_save_variables: start\n");
00439     #endif
00440     for (i = 0; i < optimize->nvariables; ++i)
00441     {
00442         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00443         fprintf (optimize->file_variables, buffer,
00444             optimize->value[simulation * optimize->
00445                 nvariables + i]);
00446         fprintf (optimize->file_variables, "%.14le\n", error);
00447         #if DEBUG
00448         fprintf (stderr, "optimize_save_variables: end\n");
00449         #endif
00450     }

```

5.17.2.16 void optimize\_step\_direction ( unsigned int *simulation* )

Function to do a step of the direction search method.

## Parameters

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

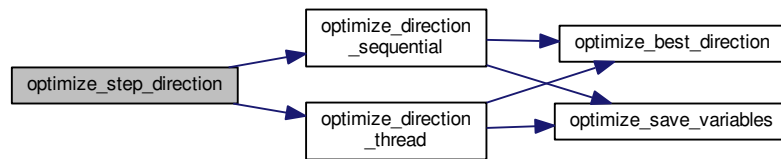
Definition at line 959 of file [optimize.c](#).

```

00960 {
00961     GThread *thread[nthreads_direction];
00962     ParallelData data[nthreads_direction];
00963     unsigned int i, j, k, b;
00964     #if DEBUG
00965         fprintf (stderr, "optimize_step_direction: start\n");
00966     #endif
00967     for (i = 0; i < optimize->nestimates; ++i)
00968     {
00969         k = (simulation + i) * optimize->nvariables;
00970         b = optimize->simulation_best[0] * optimize->
nvariables;
00971         #if DEBUG
00972             fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00973                     simulation + i, optimize->simulation_best[0]);
00974         #endif
00975         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00976         {
00977             #if DEBUG
00978                 fprintf (stderr,
00979                         "optimize_step_direction: estimate=%u best=%u%.14le\n",
00980                         i, j, optimize->value[b]);
00981             #endif
00982             optimize->value[k]
00983                 = optimize->value[b] + optimize_estimate_direction (j,
i);
00984             optimize->value[k] = fmin (fmax (optimize->value[k],
00985                                             optimize->rangeminabs[j]),
00986                                       optimize->rangemaxabs[j]);
00987             #if DEBUG
00988                 fprintf (stderr,
00989                         "optimize_step_direction: estimate=%u variable%u%.14le\n",
00990                         i, j, optimize->value[k]);
00991             #endif
00992         }
00993     }
00994     if (nthreads_direction == 1)
00995         optimize_direction_sequential (simulation);
00996     else
00997     {
00998         for (i = 0; i <= nthreads_direction; ++i)
00999         {
01000             optimize->thread_direction[i]
01001                 = simulation + optimize->nstart_direction
01002                 + i * (optimize->nend_direction - optimize->
nstart_direction)
01003                 / nthreads_direction;
01004             #if DEBUG
01005                 fprintf (stderr,
01006                         "optimize_step_direction: i=%u thread_direction=%u\n",
01007                         i, optimize->thread_direction[i]);
01008             #endif
01009         }
01010         for (i = 0; i < nthreads_direction; ++i)
01011         {
01012             data[i].thread = i;
01013             thread[i] = g_thread_new
01014                 (NULL, (void (*) ) optimize_direction_thread, &data[i]);
01015         }
01016         for (i = 0; i < nthreads_direction; ++i)
01017             g_thread_join (thread[i]);
01018     }
01019     #if DEBUG
01020         fprintf (stderr, "optimize_step_direction: end\n");
01021     #endif
01022 }

```

Here is the call graph for this function:



#### 5.17.2.17 void \* optimize\_thread ( ParallelData \* data )

Function to optimize on a thread.

##### Parameters

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

##### Returns

NULL

Definition at line 538 of file `optimize.c`.

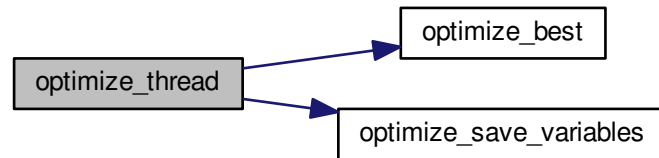
```

00539 {
00540     unsigned int i, thread;
00541     double e;
00542     #if DEBUG
00543         fprintf (stderr, "optimize_thread: start\n");
00544     #endif
00545     thread = data->thread;
00546     #if DEBUG
00547         fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00548                 optimize->thread[thread], optimize->thread[thread + 1]);
00549     #endif
00550     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00551     {
00552         e = optimize_norm (i);
00553         g_mutex_lock (mutex);
00554         optimize_best (i, e);
00555         optimize_save_variables (i, e);
00556         if (e < optimize->threshold)
00557             optimize->stop = 1;
00558         g_mutex_unlock (mutex);
00559         if (optimize->stop)
00560             break;
00561     #if DEBUG
00562         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00563     #endif
00564     }
00565     #if DEBUG
00566         fprintf (stderr, "optimize_thread: end\n");
00567     #endif
00568     g_thread_exit (NULL);
00569     return NULL;
00570 }

```



Here is the call graph for this function:



## 5.18 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 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #ifdef G_OS_WIN32
00044 #include <windows.h>
00045 #elif !defined (BSD)
00046 #include <alloca.h>
00047 #endif
00048 #if HAVE_MPI
00049 #include <mpi.h>
00050 #endif
00051 #include "genetic/genetic.h"
00052 #include "utils.h"
00053 #include "experiment.h"
00054 #include "variable.h"
00055 #include "input.h"
00056 #include "optimize.h"
00057
00058 #define DEBUG 0

```

```

00066
00070 #ifdef G_OS_WIN32
00071 #define RM "del"
00072 #else
00073 #define RM "rm"
00074 #endif
00075
00076 int ntasks;
00077 unsigned int nthreads;
00078 unsigned int nthreads_direction;
00080 GMutex mutex[1];
00081 void (*optimize_algorithm) ();
00083 double (*optimize_estimate_direction) (unsigned int variable,
00084                                         unsigned int estimate);
00086 double (*optimize_norm) (unsigned int simulation);
00088 Optimize optimize[1];
00089
00101 void
00102 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00103 {
00104     unsigned int i;
00105     char buffer[32], value[32], *buffer2, *buffer3, *content;
00106     FILE *file;
00107     gsize length;
00108     GRegex *regex;
00109
00110 #if DEBUG
00111     fprintf (stderr, "optimize_input: start\n");
00112 #endif
00113
00114     // Checking the file
00115     if (!template)
00116         goto optimize_input_end;
00117
00118     // Opening template
00119     content = g_mapped_file_get_contents (template);
00120     length = g_mapped_file_get_length (template);
00121 #if DEBUG
00122     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00123 #endif
00124     file = g_fopen (input, "w");
00125
00126     // Parsing template
00127     for (i = 0; i < optimize->nvariables; ++i)
00128     {
00129 #if DEBUG
00130         fprintf (stderr, "optimize_input: variable=%u\n", i);
00131 #endif
00132         snprintf (buffer, 32, "@variable%u@", i + 1);
00133         regex = g_regex_new (buffer, 0, 0, NULL);
00134         if (i == 0)
00135         {
00136             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00137                                                 optimize->label[i], 0, NULL);
00138 #if DEBUG
00139             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00140 #endif
00141         }
00142         else
00143         {
00144             length = strlen (buffer3);
00145             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00146                                                 optimize->label[i], 0, NULL);
00147             g_free (buffer3);
00148         }
00149         g_regex_unref (regex);
00150         length = strlen (buffer2);
00151         snprintf (buffer, 32, "@value%u@", i + 1);
00152         regex = g_regex_new (buffer, 0, 0, NULL);
00153         snprintf (value, 32, format[optimize->precision[i]],
00154                 optimize->value[simulation * optimize->nvariables + i]);
00155 #if DEBUG
00156         fprintf (stderr, "optimize_input: value=%s\n", value);
00157 #endif
00158         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00159                                           0, NULL);
00160         g_free (buffer2);
00161         g_regex_unref (regex);
00162     }
00163
00164     // Saving input file
00165     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00166     g_free (buffer3);
00167     fclose (file);
00168
00169 optimize_input_end:

```

```

00171 #if DEBUG
00172     fprintf (stderr, "optimize_input: end\n");
00173 #endif
00174     return;
00175 }
00176
00177 double
00188 optimize_parse (unsigned int simulation, unsigned int experiment)
00189 {
00190     unsigned int i;
00191     double e;
00192     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00193         *buffer3, *buffer4;
00194     FILE *file_result;
00195
00196     #if DEBUG
00197         fprintf (stderr, "optimize_parse: start\n");
00198         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00199             experiment);
00200     #endif
00201
00202     // Opening input files
00203     for (i = 0; i < optimize->ninputs; ++i)
00204     {
00205         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00206         #if DEBUG
00207             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00208         #endif
00209         optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00210     }
00211     for (; i < MAX_NINPUTS; ++i)
00212         strcpy (&input[i][0], "");
00213     #if DEBUG
00214         fprintf (stderr, "optimize_parse: parsing end\n");
00215     #endif
00216
00217     // Performing the simulation
00218     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00219     buffer2 = g_path_get_dirname (optimize->simulator);
00220     buffer3 = g_path_get_basename (optimize->simulator);
00221     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00222     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
00223         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00224         input[6], input[7], output);
00225     g_free (buffer4);
00226     g_free (buffer3);
00227     g_free (buffer2);
00228     #if DEBUG
00229         fprintf (stderr, "optimize_parse: %s\n", buffer);
00230     #endif
00231     system (buffer);
00232
00233     // Checking the objective value function
00234     if (optimize->evaluator)
00235     {
00236         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00237         buffer2 = g_path_get_dirname (optimize->evaluator);
00238         buffer3 = g_path_get_basename (optimize->evaluator);
00239         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00240         snprintf (buffer, 512, "%s\ " %s %s %s",
00241             buffer4, output, optimize->experiment[experiment], result);
00242         g_free (buffer4);
00243         g_free (buffer3);
00244         g_free (buffer2);
00245         #if DEBUG
00246             fprintf (stderr, "optimize_parse: %s\n", buffer);
00247         #endif
00248         system (buffer);
00249         file_result = g_fopen (result, "r");
00250         e = atof (fgets (buffer, 512, file_result));
00251         fclose (file_result);
00252     }
00253     else
00254     {
00255         strcpy (result, "");
00256         file_result = g_fopen (output, "r");
00257         e = atof (fgets (buffer, 512, file_result));
00258         fclose (file_result);
00259     }
00260
00261     // Removing files
00262     #if !DEBUG
00263         for (i = 0; i < optimize->ninputs; ++i)
00264         {
00265             if (optimize->file[i][0])
00266             {
00267                 snprintf (buffer, 512, RM " %s", &input[i][0]);

```

```

00268         system (buffer);
00269     }
00270 }
00271 snprintf (buffer, 512, RM " %s %s", output, result);
00272 system (buffer);
00273 #endif
00274
00275 // Processing pending events
00276 show_pending ();
00277
00278 #if DEBUG
00279 fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282 // Returning the objective function
00283 return e * optimize->weight[experiment];
00284 }
00285
00293 double
00294 optimize_norm_euclidian (unsigned int simulation)
00295 {
00296     double e, ei;
00297     unsigned int i;
00298     #if DEBUG
00299     fprintf (stderr, "optimize_norm_euclidian: start\n");
00300     #endif
00301     e = 0.;
00302     for (i = 0; i < optimize->nexperiments; ++i)
00303     {
00304         ei = optimize_parse (simulation, i);
00305         e += ei * ei;
00306     }
00307     e = sqrt (e);
00308     #if DEBUG
00309     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00310     fprintf (stderr, "optimize_norm_euclidian: end\n");
00311     #endif
00312     return e;
00313 }
00314
00322 double
00323 optimize_norm_maximum (unsigned int simulation)
00324 {
00325     double e, ei;
00326     unsigned int i;
00327     #if DEBUG
00328     fprintf (stderr, "optimize_norm_maximum: start\n");
00329     #endif
00330     e = 0.;
00331     for (i = 0; i < optimize->nexperiments; ++i)
00332     {
00333         ei = fabs (optimize_parse (simulation, i));
00334         e = fmax (e, ei);
00335     }
00336     #if DEBUG
00337     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00338     fprintf (stderr, "optimize_norm_maximum: end\n");
00339     #endif
00340     return e;
00341 }
00342
00350 double
00351 optimize_norm_p (unsigned int simulation)
00352 {
00353     double e, ei;
00354     unsigned int i;
00355     #if DEBUG
00356     fprintf (stderr, "optimize_norm_p: start\n");
00357     #endif
00358     e = 0.;
00359     for (i = 0; i < optimize->nexperiments; ++i)
00360     {
00361         ei = fabs (optimize_parse (simulation, i));
00362         e += pow (ei, optimize->p);
00363     }
00364     e = pow (e, 1. / optimize->p);
00365     #if DEBUG
00366     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00367     fprintf (stderr, "optimize_norm_p: end\n");
00368     #endif
00369     return e;
00370 }
00371
00379 double
00380 optimize_norm_taxicab (unsigned int simulation)
00381 {
00382     double e;

```

```

00383     unsigned int i;
00384     #if DEBUG
00385     fprintf (stderr, "optimize_norm_taxicab: start\n");
00386     #endif
00387     e = 0.;
00388     for (i = 0; i < optimize->nexperiments; ++i)
00389         e += fabs (optimize_parse (simulation, i));
00390     #if DEBUG
00391     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00392     fprintf (stderr, "optimize_norm_taxicab: end\n");
00393     #endif
00394     return e;
00395 }
00396
00401 void
00402 optimize_print ()
00403 {
00404     unsigned int i;
00405     char buffer[512];
00406     #if HAVE_MPI
00407     if (optimize->mpi_rank)
00408         return;
00409     #endif
00410     printf ("%s\n", gettext ("Best result"));
00411     fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
00412     printf ("error = %.15le\n", optimize->error_old[0]);
00413     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00414     for (i = 0; i < optimize->nvariables; ++i)
00415     {
00416         snprintf (buffer, 512, "%s = %s\n",
00417                 optimize->label[i], format[optimize->precision[i]]);
00418         printf (buffer, optimize->value_old[i]);
00419         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00420     }
00421     fflush (optimize->file_result);
00422 }
00423
00432 void
00433 optimize_save_variables (unsigned int simulation, double error)
00434 {
00435     unsigned int i;
00436     char buffer[64];
00437     #if DEBUG
00438     fprintf (stderr, "optimize_save_variables: start\n");
00439     #endif
00440     for (i = 0; i < optimize->nvariables; ++i)
00441     {
00442         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00443         fprintf (optimize->file_variables, buffer,
00444                 optimize->value[simulation * optimize->nvariables + i]);
00445     }
00446     fprintf (optimize->file_variables, "%.14le\n", error);
00447     #if DEBUG
00448     fprintf (stderr, "optimize_save_variables: end\n");
00449     #endif
00450 }
00451
00460 void
00461 optimize_best (unsigned int simulation, double value)
00462 {
00463     unsigned int i, j;
00464     double e;
00465     #if DEBUG
00466     fprintf (stderr, "optimize_best: start\n");
00467     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00468             optimize->nsaveds, optimize->nbest);
00469     #endif
00470     if (optimize->nsaveds < optimize->nbest
00471         || value < optimize->error_best[optimize->nsaveds - 1])
00472     {
00473         if (optimize->nsaveds < optimize->nbest)
00474             ++optimize->nsaveds;
00475         optimize->error_best[optimize->nsaveds - 1] = value;
00476         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00477         for (i = optimize->nsaveds; --i;)
00478         {
00479             if (optimize->error_best[i] < optimize->error_best[i - 1])
00480             {
00481                 j = optimize->simulation_best[i];
00482                 e = optimize->error_best[i];
00483                 optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00484                 optimize->error_best[i] = optimize->error_best[i - 1];
00485                 optimize->simulation_best[i - 1] = j;
00486                 optimize->error_best[i - 1] = e;
00487             }

```

```

00488         else
00489             break;
00490     }
00491 }
00492 #if DEBUG
00493 fprintf (stderr, "optimize_best: end\n");
00494 #endif
00495 }
00496
00501 void
00502 optimize_sequential ()
00503 {
00504     unsigned int i;
00505     double e;
00506     #if DEBUG
00507         fprintf (stderr, "optimize_sequential: start\n");
00508         fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00509             optimize->nstart, optimize->nend);
00510     #endif
00511     for (i = optimize->nstart; i < optimize->nend; ++i)
00512     {
00513         e = optimize_norm (i);
00514         optimize_best (i, e);
00515         optimize_save_variables (i, e);
00516         if (e < optimize->thresold)
00517         {
00518             optimize->stop = 1;
00519             break;
00520         }
00521     }
00522     #if DEBUG
00523         fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00524     #endif
00525     #if DEBUG
00526         fprintf (stderr, "optimize_sequential: end\n");
00527     #endif
00528 }
00529
00537 void *
00538 optimize_thread (ParallelData * data)
00539 {
00540     unsigned int i, thread;
00541     double e;
00542     #if DEBUG
00543         fprintf (stderr, "optimize_thread: start\n");
00544     #endif
00545     thread = data->thread;
00546     #if DEBUG
00547         fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00548             optimize->thread[thread], optimize->thread[thread + 1]);
00549     #endif
00550     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00551     {
00552         e = optimize_norm (i);
00553         g_mutex_lock (mutex);
00554         optimize_best (i, e);
00555         optimize_save_variables (i, e);
00556         if (e < optimize->thresold)
00557         {
00558             optimize->stop = 1;
00559             g_mutex_unlock (mutex);
00560             break;
00561         }
00562     }
00563     #if DEBUG
00564         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00565     #endif
00566     #if DEBUG
00567         fprintf (stderr, "optimize_thread: end\n");
00568     #endif
00569     g_thread_exit (NULL);
00570     return NULL;
00571 }
00572
00583 void
00584 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00585     double *error_best)
00586 {
00587     unsigned int i, j, k, s[optimize->nbest];
00588     double e[optimize->nbest];
00589     #if DEBUG
00590         fprintf (stderr, "optimize_merge: start\n");
00591     #endif
00592     i = j = k = 0;
00593     do
00594     {
00595         if (i == optimize->nsaveds)
00596         {

```

```

00597         s[k] = simulation_best[j];
00598         e[k] = error_best[j];
00599         ++j;
00600         ++k;
00601         if (j == nsaveds)
00602             break;
00603     }
00604     else if (j == nsaveds)
00605     {
00606         s[k] = optimize->simulation_best[i];
00607         e[k] = optimize->error_best[i];
00608         ++i;
00609         ++k;
00610         if (i == optimize->nsaveds)
00611             break;
00612     }
00613     else if (optimize->error_best[i] > error_best[j])
00614     {
00615         s[k] = simulation_best[j];
00616         e[k] = error_best[j];
00617         ++j;
00618         ++k;
00619     }
00620     else
00621     {
00622         s[k] = optimize->simulation_best[i];
00623         e[k] = optimize->error_best[i];
00624         ++i;
00625         ++k;
00626     }
00627 }
00628 while (k < optimize->nbest);
00629 optimize->nsaveds = k;
00630 memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00631 memcpy (optimize->error_best, e, k * sizeof (double));
00632 #if DEBUG
00633 fprintf (stderr, "optimize_merge: end\n");
00634 #endif
00635 }
00636
00641 #if HAVE_MPI
00642 void
00643 optimize_synchronise ()
00644 {
00645     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00646     double error_best[optimize->nbest];
00647     MPI_Status mpi_stat;
00648     #if DEBUG
00649     fprintf (stderr, "optimize_synchronise: start\n");
00650     #endif
00651     if (optimize->mpi_rank == 0)
00652     {
00653         for (i = 1; i < ntasks; ++i)
00654         {
00655             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00656             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00657                     MPI_COMM_WORLD, &mpi_stat);
00658             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00659                     MPI_COMM_WORLD, &mpi_stat);
00660             optimize_merge (nsaveds, simulation_best, error_best);
00661             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00662             if (stop)
00663                 optimize->stop = 1;
00664         }
00665         for (i = 1; i < ntasks; ++i)
00666             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00667     }
00668     else
00669     {
00670         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00671         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00672                 MPI_COMM_WORLD);
00673         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00674                 MPI_COMM_WORLD);
00675         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00676         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00677         if (stop)
00678             optimize->stop = 1;
00679     }
00680     #if DEBUG
00681     fprintf (stderr, "optimize_synchronise: end\n");
00682     #endif
00683 }
00684 #endif
00685
00690 void
00691 optimize_sweep ()

```

```

00692 {
00693     unsigned int i, j, k, l;
00694     double e;
00695     GThread *thread[nthreads];
00696     ParallelData data[nthreads];
00697     #if DEBUG
00698     fprintf (stderr, "optimize_sweep: start\n");
00699     #endif
00700     for (i = 0; i < optimize->nsimulations; ++i)
00701     {
00702         k = i;
00703         for (j = 0; j < optimize->nvariables; ++j)
00704         {
00705             l = k % optimize->nsweeps[j];
00706             k /= optimize->nsweeps[j];
00707             e = optimize->rangemin[j];
00708             if (optimize->nsweeps[j] > 1)
00709                 e += 1 * (optimize->rangemax[j] - optimize->rangemin[j])
00710                     / (optimize->nsweeps[j] - 1);
00711             optimize->value[i * optimize->nvariables + j] = e;
00712         }
00713     }
00714     optimize->nsaveds = 0;
00715     if (nthreads <= 1)
00716         optimize_sequential ();
00717     else
00718     {
00719         for (i = 0; i < nthreads; ++i)
00720         {
00721             data[i].thread = i;
00722             thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00723         }
00724         for (i = 0; i < nthreads; ++i)
00725             g_thread_join (thread[i]);
00726     }
00727     #if HAVE_MPI
00728     // Communicating tasks results
00729     optimize_synchronise ();
00730     #endif
00731     #if DEBUG
00732     fprintf (stderr, "optimize_sweep: end\n");
00733     #endif
00734 }
00735
00740 void
00741 optimize_MonteCarlo ()
00742 {
00743     unsigned int i, j;
00744     GThread *thread[nthreads];
00745     ParallelData data[nthreads];
00746     #if DEBUG
00747     fprintf (stderr, "optimize_MonteCarlo: start\n");
00748     #endif
00749     for (i = 0; i < optimize->nsimulations; ++i)
00750     {
00751         for (j = 0; j < optimize->nvariables; ++j)
00752             optimize->value[i * optimize->nvariables + j]
00753                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00754                     * (optimize->rangemax[j] - optimize->rangemin[j]);
00755         optimize->nsaveds = 0;
00756         if (nthreads <= 1)
00757             optimize_sequential ();
00758         else
00759         {
00760             for (i = 0; i < nthreads; ++i)
00761             {
00762                 data[i].thread = i;
00763                 thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00764             }
00765             for (i = 0; i < nthreads; ++i)
00766                 g_thread_join (thread[i]);
00767         }
00768         #if HAVE_MPI
00769         // Communicating tasks results
00770         optimize_synchronise ();
00771         #endif
00772         #if DEBUG
00773         fprintf (stderr, "optimize_MonteCarlo: end\n");
00774         #endif
00775     }
00776
00785 void
00786 optimize_best_direction (unsigned int simulation, double value)
00787 {
00788     #if DEBUG
00789     fprintf (stderr, "optimize_best_direction: start\n");
00790     fprintf (stderr,
00791         "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",

```



```

00792         simulation, value, optimize->error_best[0]);
00793 #endif
00794     if (value < optimize->error_best[0])
00795     {
00796         optimize->error_best[0] = value;
00797         optimize->simulation_best[0] = simulation;
00798 #if DEBUG
00799         fprintf (stderr,
00800             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00801             simulation, value);
00802 #endif
00803     }
00804 #if DEBUG
00805     fprintf (stderr, "optimize_best_direction: end\n");
00806 #endif
00807 }
00808
00815 void
00816 optimize_direction_sequential (unsigned int simulation)
00817 {
00818     unsigned int i, j;
00819     double e;
00820 #if DEBUG
00821     fprintf (stderr, "optimize_direction_sequential: start\n");
00822     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00823         "nend_direction=%u\n",
00824         optimize->nstart_direction, optimize->nend_direction);
00825 #endif
00826     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00827     {
00828         j = simulation + i;
00829         e = optimize_norm (j);
00830         optimize_best_direction (j, e);
00831         optimize_save_variables (j, e);
00832         if (e < optimize->threshold)
00833         {
00834             optimize->stop = 1;
00835             break;
00836         }
00837 #if DEBUG
00838         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00839 #endif
00840     }
00841 #if DEBUG
00842     fprintf (stderr, "optimize_direction_sequential: end\n");
00843 #endif
00844 }
00845
00853 void *
00854 optimize_direction_thread (ParallelData * data)
00855 {
00856     unsigned int i, thread;
00857     double e;
00858 #if DEBUG
00859     fprintf (stderr, "optimize_direction_thread: start\n");
00860 #endif
00861     thread = data->thread;
00862 #if DEBUG
00863     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00864         thread,
00865         optimize->thread_direction[thread],
00866         optimize->thread_direction[thread + 1]);
00867 #endif
00868     for (i = optimize->thread_direction[thread];
00869         i < optimize->thread_direction[thread + 1]; ++i)
00870     {
00871         e = optimize_norm (i);
00872         g_mutex_lock (mutex);
00873         optimize_best_direction (i, e);
00874         optimize_save_variables (i, e);
00875         if (e < optimize->threshold)
00876             optimize->stop = 1;
00877         g_mutex_unlock (mutex);
00878         if (optimize->stop)
00879             break;
00880 #if DEBUG
00881         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00882 #endif
00883     }
00884 #if DEBUG
00885     fprintf (stderr, "optimize_direction_thread: end\n");
00886 #endif
00887     g_thread_exit (NULL);
00888     return NULL;
00889 }
00890
00900 double

```

```

00901 optimize_estimate_direction_random (unsigned int variable,
00902                                     unsigned int estimate)
00903 {
00904     double x;
00905     #if DEBUG
00906     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907     #endif
00908     x = optimize->direction[variable]
00909         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00910     #if DEBUG
00911     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00912             variable, x);
00913     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00914     #endif
00915     return x;
00916 }
00917
00927 double
00928 optimize_estimate_direction_coordinates (unsigned int variable,
00929                                         unsigned int estimate)
00930 {
00931     double x;
00932     #if DEBUG
00933     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00934     #endif
00935     x = optimize->direction[variable];
00936     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00937     {
00938         if (estimate & 1)
00939             x += optimize->step[variable];
00940         else
00941             x -= optimize->step[variable];
00942     }
00943     #if DEBUG
00944     fprintf (stderr,
00945             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00946             variable, x);
00947     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00948     #endif
00949     return x;
00950 }
00951
00958 void
00959 optimize_step_direction (unsigned int simulation)
00960 {
00961     GThread *thread[nthreads_direction];
00962     ParallelData data[nthreads_direction];
00963     unsigned int i, j, k, b;
00964     #if DEBUG
00965     fprintf (stderr, "optimize_step_direction: start\n");
00966     #endif
00967     for (i = 0; i < optimize->nestimates; ++i)
00968     {
00969         k = (simulation + i) * optimize->nvariables;
00970         b = optimize->simulation_best[0] * optimize->nvariables;
00971         #if DEBUG
00972         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00973                 simulation + i, optimize->simulation_best[0]);
00974         #endif
00975         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00976         {
00977             #if DEBUG
00978             fprintf (stderr,
00979                     "optimize_step_direction: estimate=%u best%u=%%.14le\n",
00980                     i, j, optimize->value[b]);
00981             #endif
00982             optimize->value[k]
00983                 = optimize->value[b] + optimize_estimate_direction (j, i);
00984             optimize->value[k] = fmin (fmax (optimize->value[k],
00985                                         optimize->rangeminabs[j]),
00986                                     optimize->rangemaxabs[j]);
00987             #if DEBUG
00988             fprintf (stderr,
00989                     "optimize_step_direction: estimate=%u variable%u=%%.14le\n",
00990                     i, j, optimize->value[k]);
00991             #endif
00992         }
00993     }
00994     if (nthreads_direction == 1)
00995         optimize_direction_sequential (simulation);
00996     else
00997     {
00998         for (i = 0; i <= nthreads_direction; ++i)
00999         {
01000             optimize->thread_direction[i]
01001                 = simulation + optimize->nstart_direction
01002                 + i * (optimize->nend_direction - optimize->

```

```

nstart_direction)
01003 / nthreads_direction;
01004 #if DEBUG
01005     fprintf (stderr,
01006             "optimize_step_direction: i=%u thread_direction=%u\n",
01007             i, optimize->thread_direction[i]);
01008 #endif
01009 }
01010 for (i = 0; i < nthreads_direction; ++i)
01011 {
01012     data[i].thread = i;
01013     thread[i] = g_thread_new
01014         (NULL, (void (*) ) optimize_direction_thread, &data[i]);
01015 }
01016 for (i = 0; i < nthreads_direction; ++i)
01017     g_thread_join (thread[i]);
01018 }
01019 #if DEBUG
01020     fprintf (stderr, "optimize_step_direction: end\n");
01021 #endif
01022 }
01023
01024 void
01025 optimize_direction ()
01026 {
01027     unsigned int i, j, k, b, s, adjust;
01028     #if DEBUG
01029         fprintf (stderr, "optimize_direction: start\n");
01030     #endif
01031     for (i = 0; i < optimize->nvariables; ++i)
01032         optimize->direction[i] = 0.;
01033     b = optimize->simulation_best[0] * optimize->nvariables;
01034     s = optimize->nsimulations;
01035     adjust = 1;
01036     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01037     {
01038         #if DEBUG
01039             fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01040                     i, optimize->simulation_best[0]);
01041         #endif
01042         optimize_step_direction (s);
01043         k = optimize->simulation_best[0] * optimize->nvariables;
01044         #if DEBUG
01045             fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01046                     i, optimize->simulation_best[0]);
01047         #endif
01048         if (k == b)
01049         {
01050             if (adjust)
01051             {
01052                 for (j = 0; j < optimize->nvariables; ++j)
01053                     optimize->step[j] *= 0.5;
01054                 for (j = 0; j < optimize->nvariables; ++j)
01055                     optimize->direction[j] = 0.;
01056                 adjust = 1;
01057             }
01058             else
01059             {
01060                 for (j = 0; j < optimize->nvariables; ++j)
01061                 {
01062                     #if DEBUG
01063                         fprintf (stderr,
01064                                 "optimize_direction: best%u=%.14le old%u=%.14le\n",
01065                                 j, optimize->value[k + j], j, optimize->value[b + j]);
01066                     #endif
01067                     optimize->direction[j]
01068                         = (1. - optimize->relaxation) * optimize->direction[j]
01069                         + optimize->relaxation
01070                         * (optimize->value[k + j] - optimize->value[b + j]);
01071                     #if DEBUG
01072                         fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01073                                 j, optimize->direction[j]);
01074                     #endif
01075                 }
01076                 adjust = 0;
01077             }
01078         }
01079         #if DEBUG
01080             fprintf (stderr, "optimize_direction: end\n");
01081         #endif
01082     }
01083
01084     double
01085     optimize_genetic_objective (Entity * entity)
01086     {
01087         unsigned int j;
01088         double objective;
01089         char buffer[64];

```

```

01100 #if DEBUG
01101 fprintf (stderr, "optimize_genetic_objective: start\n");
01102 #endif
01103 for (j = 0; j < optimize->nvariables; ++j)
01104 {
01105     optimize->value[entity->id * optimize->nvariables + j]
01106         = genetic_get_variable (entity, optimize->genetic_variable + j);
01107 }
01108 objective = optimize_norm (entity->id);
01109 g_mutex_lock (mutex);
01110 for (j = 0; j < optimize->nvariables; ++j)
01111 {
01112     snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01113     fprintf (optimize->file_variables, buffer,
01114             genetic_get_variable (entity, optimize->genetic_variable + j));
01115 }
01116 fprintf (optimize->file_variables, "%.14le\n", objective);
01117 g_mutex_unlock (mutex);
01118 #if DEBUG
01119 fprintf (stderr, "optimize_genetic_objective: end\n");
01120 #endif
01121 return objective;
01122 }
01123
01124 void
01125 optimize_genetic ()
01126 {
01127     char *best_genome;
01128     double best_objective, *best_variable;
01129 #if DEBUG
01130     fprintf (stderr, "optimize_genetic: start\n");
01131     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01132             nthreads);
01133     fprintf (stderr,
01134             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01135             optimize->nvariables, optimize->nsimulations, optimize->
01136             niterations);
01137     fprintf (stderr,
01138             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01139             optimize->mutation_ratio, optimize->reproduction_ratio,
01140             optimize->adaptation_ratio);
01141 #endif
01142     genetic_algorithm_default (optimize->nvariables,
01143                               optimize->genetic_variable,
01144                               optimize->nsimulations,
01145                               optimize->niterations,
01146                               optimize->mutation_ratio,
01147                               optimize->reproduction_ratio,
01148                               optimize->adaptation_ratio,
01149                               optimize->threshold,
01150                               &optimize_genetic_objective,
01151                               &best_genome, &best_variable, &best_objective);
01152 #if DEBUG
01153     fprintf (stderr, "optimize_genetic: the best\n");
01154 #endif
01155     optimize->error_old = (double *) g_malloc (sizeof (double));
01156     optimize->value_old
01157         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01158     optimize->error_old[0] = best_objective;
01159     memcpy (optimize->value_old, best_variable,
01160            optimize->nvariables * sizeof (double));
01161     g_free (best_genome);
01162     g_free (best_variable);
01163     optimize_print ();
01164 #if DEBUG
01165     fprintf (stderr, "optimize_genetic: end\n");
01166 #endif
01167 }
01168
01169 void
01170 optimize_save_old ()
01171 {
01172     unsigned int i, j;
01173 #if DEBUG
01174     fprintf (stderr, "optimize_save_old: start\n");
01175     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01176 #endif
01177     memcpy (optimize->error_old, optimize->error_best,
01178            optimize->nbest * sizeof (double));
01179     for (i = 0; i < optimize->nbest; ++i)
01180     {
01181         j = optimize->simulation_best[i];
01182 #if DEBUG
01183         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01184 #endif
01185         memcpy (optimize->value_old + i * optimize->nvariables,
01186                optimize->value + j * optimize->nvariables,

```

```

01194         optimize->nvariables * sizeof (double));
01195     }
01196     #if DEBUG
01197     for (i = 0; i < optimize->nvariables; ++i)
01198         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01199                 i, optimize->value_old[i]);
01200     fprintf (stderr, "optimize_save_old: end\n");
01201     #endif
01202 }
01203
01204 void
01205 optimize_merge_old ()
01206 {
01207     unsigned int i, j, k;
01208     double v[optimize->nbest * optimize->nvariables], e[optimize->
nbest],
01209           *enew, *eold;
01210     #if DEBUG
01211     fprintf (stderr, "optimize_merge_old: start\n");
01212     #endif
01213     enew = optimize->error_best;
01214     eold = optimize->error_old;
01215     i = j = k = 0;
01216     do
01217     {
01218         if (*enew < *eold)
01219         {
01220             memcpy (v + k * optimize->nvariables,
01221                     optimize->value
01222                     + optimize->simulation_best[i] * optimize->
nvariables,
01223                     optimize->nvariables * sizeof (double));
01224             e[k] = *enew;
01225             ++k;
01226             ++enew;
01227             ++i;
01228         }
01229         else
01230         {
01231             memcpy (v + k * optimize->nvariables,
01232                     optimize->value_old + j * optimize->nvariables,
01233                     optimize->nvariables * sizeof (double));
01234             e[k] = *eold;
01235             ++k;
01236             ++eold;
01237             ++j;
01238         }
01239     }
01240     while (k < optimize->nbest);
01241     memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01242     memcpy (optimize->error_old, e, k * sizeof (double));
01243     #if DEBUG
01244     fprintf (stderr, "optimize_merge_old: end\n");
01245     #endif
01246 }
01247
01248 void
01249 optimize_refine ()
01250 {
01251     unsigned int i, j;
01252     double d;
01253     #if HAVE_MPI
01254     MPI_Status mpi_stat;
01255     #endif
01256     #if DEBUG
01257     fprintf (stderr, "optimize_refine: start\n");
01258     #endif
01259     #if HAVE_MPI
01260     if (!optimize->mpi_rank)
01261     {
01262         for (j = 0; j < optimize->nvariables; ++j)
01263         {
01264             optimize->rangemin[j] = optimize->rangemax[j]
= optimize->value_old[j];
01265         }
01266         for (i = 0; ++i < optimize->nbest;)
01267         {
01268             for (j = 0; j < optimize->nvariables; ++j)
01269             {
01270                 optimize->rangemin[j]
= fmin (optimize->rangemin[j],
01271          optimize->value_old[i * optimize->nvariables + j]);
01272                 optimize->rangemax[j]
= fmax (optimize->rangemax[j],
01273          optimize->value_old[i * optimize->nvariables + j]);
01274             }
01275         }
01276     }
01277 }

```

```

01289     }
01290     for (j = 0; j < optimize->nvariables; ++j)
01291     {
01292         d = optimize->tolerance
01293         * (optimize->rangemax[j] - optimize->rangemin[j]);
01294         switch (optimize->algorithm)
01295         {
01296             case ALGORITHM_MONTE_CARLO:
01297                 d *= 0.5;
01298                 break;
01299             default:
01300                 if (optimize->nsweeps[j] > 1)
01301                     d /= optimize->nsweeps[j] - 1;
01302                 else
01303                     d = 0.;
01304         }
01305         optimize->rangemin[j] -= d;
01306         optimize->rangemin[j]
01307         = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01308         optimize->rangemax[j] += d;
01309         optimize->rangemax[j]
01310         = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01311         printf ("%s min=%lg max=%lg\n", optimize->label[j],
01312             optimize->rangemin[j], optimize->rangemax[j]);
01313         fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01314             optimize->label[j], optimize->rangemin[j],
01315             optimize->rangemax[j]);
01316     }
01317     #if HAVE_MPI
01318     for (i = 1; i < ntasks; ++i)
01319     {
01320         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01321             1, MPI_COMM_WORLD);
01322         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01323             1, MPI_COMM_WORLD);
01324     }
01325     }
01326     else
01327     {
01328         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01329             MPI_COMM_WORLD, &mpi_stat);
01330         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01331             MPI_COMM_WORLD, &mpi_stat);
01332     }
01333     #endif
01334     #if DEBUG
01335     fprintf (stderr, "optimize_refine: end\n");
01336     #endif
01337 }
01338
01339 void
01340 optimize_step ()
01341 {
01342     #if DEBUG
01343     fprintf (stderr, "optimize_step: start\n");
01344     #endif
01345     optimize_algorithm ();
01346     if (optimize->nsteps)
01347         optimize_direction ();
01348     #if DEBUG
01349     fprintf (stderr, "optimize_step: end\n");
01350     #endif
01351 }
01352
01353 void
01354 optimize_iterate ()
01355 {
01356     unsigned int i;
01357     #if DEBUG
01358     fprintf (stderr, "optimize_iterate: start\n");
01359     #endif
01360     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01361     optimize->value_old = (double *)
01362         g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
01363     optimize_step ();
01364     optimize_save_old ();
01365     optimize_refine ();
01366     optimize_print ();
01367     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01368     {
01369         optimize_step ();
01370         optimize_merge_old ();
01371         optimize_refine ();
01372         optimize_print ();
01373     }
01374     #if DEBUG
01375     fprintf (stderr, "optimize_iterate: end\n");
01376     #endif
01377 }

```

```

01384 #endif
01385 }
01386
01391 void
01392 optimize_free ()
01393 {
01394     unsigned int i, j;
01395     #if DEBUG
01396     fprintf (stderr, "optimize_free: start\n");
01397     #endif
01398     for (j = 0; j < optimize->ninputs; ++j)
01399     {
01400         for (i = 0; i < optimize->nexperiments; ++i)
01401             g_mapped_file_unref (optimize->file[j][i]);
01402         g_free (optimize->file[j]);
01403     }
01404     g_free (optimize->error_old);
01405     g_free (optimize->value_old);
01406     g_free (optimize->value);
01407     g_free (optimize->genetic_variable);
01408     #if DEBUG
01409     fprintf (stderr, "optimize_free: end\n");
01410     #endif
01411 }
01412
01417 void
01418 optimize_open ()
01419 {
01420     GTimeZone *tz;
01421     GDateTime *t0, *t;
01422     unsigned int i, j, *nbits;
01423
01424     #if DEBUG
01425     char *buffer;
01426     fprintf (stderr, "optimize_open: start\n");
01427     #endif
01428
01429     // Getting initial time
01430     #if DEBUG
01431     fprintf (stderr, "optimize_open: getting initial time\n");
01432     #endif
01433     tz = g_time_zone_new_utc ();
01434     t0 = g_date_time_new_now (tz);
01435
01436     // Obtaining and initing the pseudo-random numbers generator seed
01437     #if DEBUG
01438     fprintf (stderr, "optimize_open: getting initial seed\n");
01439     #endif
01440     optimize->seed = input->seed;
01441     gsl_rng_set (optimize->rng, optimize->seed);
01442
01443     // Replacing the working directory
01444     #if DEBUG
01445     fprintf (stderr, "optimize_open: replacing the working directory\n");
01446     #endif
01447     g_chdir (input->directory);
01448
01449     // Getting results file names
01450     optimize->result = input->result;
01451     optimize->variables = input->variables;
01452
01453     // Obtaining the simulator file
01454     optimize->simulator = input->simulator;
01455
01456     // Obtaining the evaluator file
01457     optimize->evaluator = input->evaluator;
01458
01459     // Reading the algorithm
01460     optimize->algorithm = input->algorithm;
01461     switch (optimize->algorithm)
01462     {
01463         case ALGORITHM_MONTE_CARLO:
01464             optimize_algorithm = optimize_MonteCarlo;
01465             break;
01466         case ALGORITHM_SWEEP:
01467             optimize_algorithm = optimize_sweep;
01468             break;
01469         default:
01470             optimize_algorithm = optimize_genetic;
01471             optimize->mutation_ratio = input->mutation_ratio;
01472             optimize->reproduction_ratio = input->reproduction_ratio;
01473             optimize->adaptation_ratio = input->adaptation_ratio;
01474     }
01475     optimize->nvariables = input->nvariables;
01476     optimize->nsimulations = input->nsimulations;
01477     optimize->niterations = input->niterations;
01478     optimize->nbest = input->nbest;

```

```

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

```



```

rangemaxabs;
01563     optimize->precision[i] = input->variable[i].precision;
01564     optimize->step[i] = input->variable[i].step;
01565     optimize->nsweeps[i] = input->variable[i].nsweeps;
01566     optimize->nbits[i] = input->variable[i].nbits;
01567 }
01568 if (input->algorithm == ALGORITHM_SWEEP)
01569 {
01570     optimize->nsimulations = 1;
01571     for (i = 0; i < input->nvariables; ++i)
01572     {
01573         if (input->algorithm == ALGORITHM_SWEEP)
01574         {
01575             optimize->nsimulations *= optimize->nsweeps[i];
01576 #if DEBUG
01577             fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01578                     optimize->nsweeps[i], optimize->nsimulations);
01579 #endif
01580         }
01581     }
01582 }
01583 if (optimize->nsteps)
01584     optimize->direction
01585     = (double *) alloca (optimize->nvariables * sizeof (double));
01586 // Setting error norm
01587 switch (input->norm)
01588 {
01589     case ERROR_NORM_EUCLIDIAN:
01590         optimize_norm = optimize_norm_euclidian;
01591         break;
01592     case ERROR_NORM_MAXIMUM:
01593         optimize_norm = optimize_norm_maximum;
01594         break;
01595     case ERROR_NORM_P:
01596         optimize_norm = optimize_norm_p;
01597         optimize->p = input->p;
01598         break;
01599     default:
01600         optimize_norm = optimize_norm_taxicab;
01601 }
01602 // Allocating values
01603 #if DEBUG
01604     fprintf (stderr, "optimize_open: allocating variables\n");
01605     fprintf (stderr, "optimize_open: nvariables=%u\n", optimize->nvariables);
01606 #endif
01607 optimize->genetic_variable = NULL;
01608 if (optimize->algorithm == ALGORITHM_GENETIC)
01609 {
01610     optimize->genetic_variable = (GeneticVariable *)
01611     g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01612     for (i = 0; i < optimize->nvariables; ++i)
01613     {
01614         #if DEBUG
01615             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01616                     i, optimize->rangemin[i], optimize->rangemax[i], nbits[i]);
01617         #endif
01618         optimize->genetic_variable[i].minimum = optimize->
01619         rangemin[i];
01620         optimize->genetic_variable[i].maximum = optimize->
01621         rangemax[i];
01622         optimize->genetic_variable[i].nbits = nbits[i];
01623     }
01624 }
01625 #if DEBUG
01626     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01627             optimize->nvariables, optimize->nsimulations);
01628 #endif
01629 optimize->value = (double *)
01630 g_malloc ((optimize->nsimulations
01631 + optimize->nestimates * optimize->nsteps)
01632 * optimize->nvariables * sizeof (double));
01633 // Calculating simulations to perform for each task
01634 #if HAVE_MPI
01635 #if DEBUG
01636     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01637             optimize->mpi_rank, ntasks);
01638 #endif
01639 optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01640 ntasks;
01641 optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01642 ntasks;
01643 if (optimize->nsteps)
01644 {
01645     optimize->nstart_direction

```

```

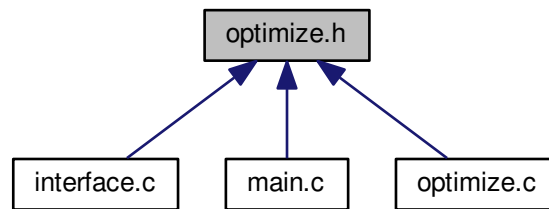
01645         = optimize->mpi_rank * optimize->nestimates / ntasks;
01646         optimize->nend_direction
01647         = (1 + optimize->mpi_rank) * optimize->nestimates /
           ntasks;
01648     }
01649 #else
01650     optimize->nstart = 0;
01651     optimize->nend = optimize->nsimulations;
01652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

```

## 5.19 optimize.h File Reference

Header file to define the optimization functions.

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



## Data Structures

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

## Functions

- void [optimize\\_input](#) (unsigned int simulation, char \*input, GMappedFile \*template)  
*Function to write the simulation input file.*
- double [optimize\\_parse](#) (unsigned int simulation, unsigned int experiment)  
*Function to parse input files, simulating and calculating the \ objective function.*
- double [optimize\\_norm\\_euclidian](#) (unsigned int simulation)  
*Function to calculate the Euclidian error norm.*
- double [optimize\\_norm\\_maximum](#) (unsigned int simulation)  
*Function to calculate the maximum error norm.*
- double [optimize\\_norm\\_p](#) (unsigned int simulation)  
*Function to calculate the P error norm.*
- double [optimize\\_norm\\_taxicab](#) (unsigned int simulation)  
*Function to calculate the taxicab error norm.*
- void [optimize\\_print](#) ()  
*Function to print the results.*
- void [optimize\\_save\\_variables](#) (unsigned int simulation, double error)  
*Function to save in a file the variables and the error.*
- void [optimize\\_best](#) (unsigned int simulation, double value)  
*Function to save the best simulations.*
- void [optimize\\_sequential](#) ()  
*Function to optimize sequentially.*
- void \* [optimize\\_thread](#) ([ParallelData](#) \*data)  
*Function to optimize on a thread.*
- void [optimize\\_merge](#) (unsigned int nsaveds, unsigned int \*simulation\_best, double \*error\_best)  
*Function to merge the 2 optimization results.*
- void [optimize\\_synchronise](#) ()  
*Function to synchronise the optimization results of MPI tasks.*

- void [optimize\\_sweep](#) ()  
*Function to optimize with the sweep algorithm.*
- void [optimize\\_MonteCarlo](#) ()  
*Function to optimize with the Monte-Carlo algorithm.*
- void [optimize\\_best\\_direction](#) (unsigned int simulation, double value)  
*Function to save the best simulation in a direction search method.*
- void **optimize\_direction\_sequential** ()
- void \* [optimize\\_direction\\_thread](#) (ParallelData \*data)  
*Function to estimate the direction search on a thread.*
- double [optimize\\_estimate\\_direction\\_random](#) (unsigned int variable, unsigned int estimate)  
*Function to estimate a component of the direction search vector.*
- double [optimize\\_estimate\\_direction\\_coordinates](#) (unsigned int variable, unsigned int estimate)  
*Function to estimate a component of the direction search vector.*
- void [optimize\\_step\\_direction](#) (unsigned int simulation)  
*Function to do a step of the direction search method.*
- void [optimize\\_direction](#) ()  
*Function to optimize with a direction search method.*
- double [optimize\\_genetic\\_objective](#) (Entity \*entity)  
*Function to calculate the objective function of an entity.*
- void [optimize\\_genetic](#) ()  
*Function to optimize with the genetic algorithm.*
- void [optimize\\_save\\_old](#) ()  
*Function to save the best results on iterative methods.*
- void [optimize\\_merge\\_old](#) ()  
*Function to merge the best results with the previous step best results on iterative methods.*
- void [optimize\\_refine](#) ()  
*Function to refine the search ranges of the variables in iterative algorithms.*
- void [optimize\\_step](#) ()  
*Function to do a step of the iterative algorithm.*
- void [optimize\\_iterate](#) ()  
*Function to iterate the algorithm.*
- void [optimize\\_free](#) ()  
*Function to free the memory used by the [Optimize](#) struct.*
- void [optimize\\_open](#) ()  
*Function to open and perform a optimization.*

## Variables

- int [ntasks](#)  
*Number of tasks.*
- unsigned int [nthreads](#)  
*Number of threads.*
- unsigned int [nthreads\\_direction](#)  
*Number of threads for the direction search method.*
- GMutex [mutex](#) [1]  
*Mutex struct.*
- void(\* [optimize\\_algorithm](#) )()  
*Pointer to the function to perform a optimization algorithm step.*
- double(\* [optimize\\_estimate\\_direction](#) )(unsigned int variable, unsigned int estimate)  
*Pointer to the function to estimate the direction.*

- `double(* optimize\_norm )(unsigned int simulation)`  
*Pointer to the error norm function.*
- `Optimize optimize [1]`  
*Optimization data.*
- `const xmlChar * result\_name`  
*Name of the result file.*
- `const xmlChar * variables\_name`  
*Name of the variables file.*

### 5.19.1 Detailed Description

Header file to define the optimization functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.19.2 Function Documentation

#### 5.19.2.1 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 461 of file [optimize.c](#).

```

00462 {
00463     unsigned int i, j;
00464     double e;
00465     #if DEBUG
00466         fprintf (stderr, "optimize_best: start\n");
00467         fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00468                 optimize->nsaveds, optimize->nbest);
00469     #endif
00470     if (optimize->nsaveds < optimize->nbest
00471         || value < optimize->error_best[optimize->nsaveds - 1])
00472     {
00473         if (optimize->nsaveds < optimize->nbest)
00474             ++optimize->nsaveds;
00475         optimize->error_best[optimize->nsaveds - 1] = value;
00476         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00477         for (i = optimize->nsaveds; --i;)
00478         {
00479             if (optimize->error_best[i] < optimize->
00480                 error_best[i - 1])
00481             {
00482                 j = optimize->simulation_best[i];
00483                 e = optimize->error_best[i];
00484                 optimize->simulation_best[i] = optimize->
00485                     simulation_best[i - 1];
00486                 optimize->error_best[i] = optimize->
00487                     error_best[i - 1];
00488                 optimize->simulation_best[i - 1] = j;
00489                 optimize->error_best[i - 1] = e;
00490             }
00491         }
00492     }
00493     else

```

```

00489         break;
00490     }
00491 }
00492 #if DEBUG
00493 fprintf (stderr, "optimize_best: end\n");
00494 #endif
00495 }

```

### 5.19.2.2 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 786 of file [optimize.c](#).

```

00787 {
00788 #if DEBUG
00789     fprintf (stderr, "optimize_best_direction: start\n");
00790     fprintf (stderr,
00791             "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00792             simulation, value, optimize->error_best[0]);
00793 #endif
00794     if (value < optimize->error_best[0])
00795     {
00796         optimize->error_best[0] = value;
00797         optimize->simulation_best[0] = simulation;
00798     #if DEBUG
00799         fprintf (stderr,
00800                 "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00801                 simulation, value);
00802     #endif
00803     }
00804 #if DEBUG
00805     fprintf (stderr, "optimize_best_direction: end\n");
00806 #endif
00807 }

```

### 5.19.2.3 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 854 of file [optimize.c](#).

```

00855 {
00856     unsigned int i, thread;
00857     double e;
00858     #if DEBUG
00859         fprintf (stderr, "optimize_direction_thread: start\n");
00860     #endif
00861     thread = data->thread;
00862     #if DEBUG
00863         fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00864                 thread,
00865                 optimize->thread_direction[thread],
00866                 optimize->thread_direction[thread + 1]);
00867     #endif
00868     for (i = optimize->thread_direction[thread];
00869          i < optimize->thread_direction[thread + 1]; ++i)
00870     {

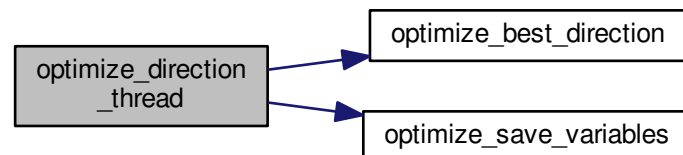
```

```

00871     e = optimize_norm (i);
00872     g_mutex_lock (mutex);
00873     optimize_best_direction (i, e);
00874     optimize_save_variables (i, e);
00875     if (e < optimize->thresold)
00876         optimize->stop = 1;
00877     g_mutex_unlock (mutex);
00878     if (optimize->stop)
00879         break;
00880 #if DEBUG
00881     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00882 #endif
00883 }
00884 #if DEBUG
00885     fprintf (stderr, "optimize_direction_thread: end\n");
00886 #endif
00887     g_thread_exit (NULL);
00888     return NULL;
00889 }

```

Here is the call graph for this function:



#### 5.19.2.4 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 928 of file [optimize.c](#).

```

00930 {
00931     double x;
00932     #if DEBUG
00933     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00934     #endif
00935     x = optimize->direction[variable];
00936     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00937     {
00938         if (estimate & 1)
00939             x += optimize->step[variable];
00940         else
00941             x -= optimize->step[variable];
00942     }
00943     #if DEBUG
00944     fprintf (stderr,
00945             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00946             variable, x);
00947     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00948     #endif
00949     return x;
00950 }

```

5.19.2.5 `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 901 of file `optimize.c`.

```

00903 {
00904     double x;
00905     #if DEBUG
00906     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907     #endif
00908     x = optimize->direction[variable]
00909         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00910         step[variable];
00911     #if DEBUG
00912     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00913             variable, x);
00914     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00915     #endif
00916     return x;
00917 }
```

### 5.19.2.6 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 1095 of file `optimize.c`.

```

01096 {
01097     unsigned int j;
01098     double objective;
01099     char buffer[64];
01100     #if DEBUG
01101     fprintf (stderr, "optimize_genetic_objective: start\n");
01102     #endif
01103     for (j = 0; j < optimize->nvariables; ++j)
01104     {
01105         optimize->value[entity->id * optimize->nvariables + j]
01106             = genetic_get_variable (entity, optimize->genetic_variable + j);
01107     }
01108     objective = optimize_norm (entity->id);
01109     g_mutex_lock (mutex);
01110     for (j = 0; j < optimize->nvariables; ++j)
01111     {
01112         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01113         fprintf (optimize->file_variables, buffer,
01114                 genetic_get_variable (entity, optimize->genetic_variable + j));
01115     }
01116     fprintf (optimize->file_variables, "%.14le\n", objective);
01117     g_mutex_unlock (mutex);
01118     #if DEBUG
01119     fprintf (stderr, "optimize_genetic_objective: end\n");
01120     #endif
01121     return objective;
01122 }
```

### 5.19.2.7 void optimize\_input ( unsigned int simulation, char \* input, GMappedFile \* template )

Function to write the simulation input file.

## Parameters

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

Definition at line 102 of file [optimize.c](#).

```

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

```

5.19.2.8 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 584 of file [optimize.c](#).

```

00586 {
00587     unsigned int i, j, k, s[optimize->nbest];
00588     double e[optimize->nbest];
00589     #if DEBUG
00590     fprintf (stderr, "optimize_merge: start\n");
00591     #endif
00592     i = j = k = 0;
00593     do
00594     {
00595         if (i == optimize->nsaveds)
00596         {
00597             s[k] = simulation_best[j];
00598             e[k] = error_best[j];
00599             ++j;
00600             ++k;
00601             if (j == nsaveds)
00602                 break;
00603         }
00604         else if (j == nsaveds)
00605         {
00606             s[k] = optimize->simulation_best[i];
00607             e[k] = optimize->error_best[i];
00608             ++i;
00609             ++k;
00610             if (i == optimize->nsaveds)
00611                 break;
00612         }
00613         else if (optimize->error_best[i] > error_best[j])
00614         {
00615             s[k] = simulation_best[j];
00616             e[k] = error_best[j];
00617             ++j;
00618             ++k;
00619         }
00620         else
00621         {
00622             s[k] = optimize->simulation_best[i];
00623             e[k] = optimize->error_best[i];
00624             ++i;
00625             ++k;
00626         }
00627     }
00628     while (k < optimize->nbest);
00629     optimize->nsaveds = k;
00630     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00631     memcpy (optimize->error_best, e, k * sizeof (double));
00632     #if DEBUG
00633     fprintf (stderr, "optimize_merge: end\n");
00634     #endif
00635 }

```

### 5.19.2.9 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 294 of file [optimize.c](#).

```

00295 {
00296     double e, ei;

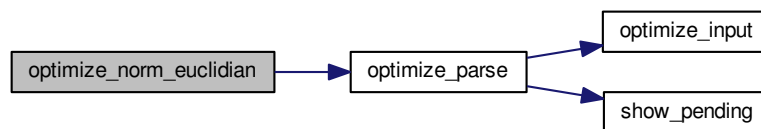
```

```

00297 unsigned int i;
00298 #if DEBUG
00299 fprintf (stderr, "optimize_norm_euclidian: start\n");
00300 #endif
00301 e = 0.;
00302 for (i = 0; i < optimize->nexperiments; ++i)
00303 {
00304     ei = optimize_parse (simulation, i);
00305     e += ei * ei;
00306 }
00307 e = sqrt (e);
00308 #if DEBUG
00309 fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00310 fprintf (stderr, "optimize_norm_euclidian: end\n");
00311 #endif
00312 return e;
00313 }

```

Here is the call graph for this function:



#### 5.19.2.10 double optimize\_norm\_maximum ( unsigned int *simulation* )

Function to calculate the maximum error norm.

##### Parameters

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

##### Returns

Maximum error norm.

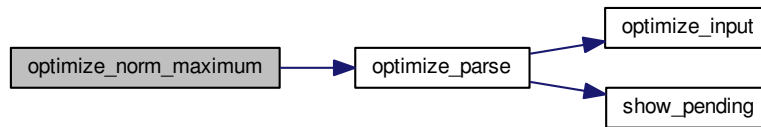
Definition at line 323 of file [optimize.c](#).

```

00324 {
00325     double e, ei;
00326     unsigned int i;
00327     #if DEBUG
00328     fprintf (stderr, "optimize_norm_maximum: start\n");
00329     #endif
00330     e = 0.;
00331     for (i = 0; i < optimize->nexperiments; ++i)
00332     {
00333         ei = fabs (optimize_parse (simulation, i));
00334         e = fmax (e, ei);
00335     }
00336     #if DEBUG
00337     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00338     fprintf (stderr, "optimize_norm_maximum: end\n");
00339     #endif
00340     return e;
00341 }

```

Here is the call graph for this function:



#### 5.19.2.11 double optimize\_norm\_p ( unsigned int *simulation* )

Function to calculate the P error norm.

##### Parameters

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

##### Returns

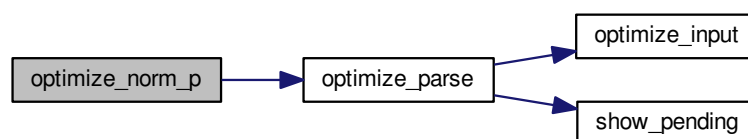
P error norm.

Definition at line 351 of file [optimize.c](#).

```

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

Here is the call graph for this function:



5.19.2.12 double optimize\_norm\_taxicab ( unsigned int *simulation* )

Function to calculate the taxicab error norm.

## Parameters

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

## Returns

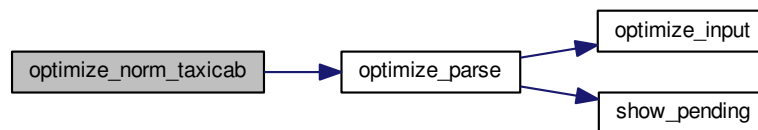
Taxicab error norm.

Definition at line 380 of file [optimize.c](#).

```

00381 {
00382     double e;
00383     unsigned int i;
00384     #if DEBUG
00385     fprintf (stderr, "optimize_norm_taxicab: start\n");
00386     #endif
00387     e = 0.;
00388     for (i = 0; i < optimize->nexperiments; ++i)
00389         e += fabs (optimize_parse (simulation, i));
00390     #if DEBUG
00391     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00392     fprintf (stderr, "optimize_norm_taxicab: end\n");
00393     #endif
00394     return e;
00395 }
```

Here is the call graph for this function:



### 5.19.2.13 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>	<a href="#">Experiment</a> number.

## Returns

Objective function value.

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

```

00189 {
00190     unsigned int i;
00191     double e;
00192     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00193         *buffer3, *buffer4;
00194     FILE *file_result;
00195
00196     #if DEBUG
00197     fprintf (stderr, "optimize_parse: start\n");
00198     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00199         experiment);
```

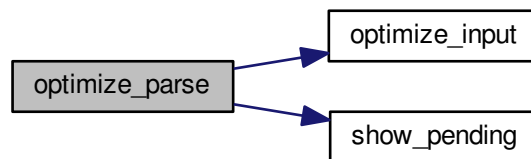


```

00200 #endif
00201
00202 // Opening input files
00203 for (i = 0; i < optimize->ninputs; ++i)
00204 {
00205     snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00206 #if DEBUG
00207     fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00208 #endif
00209     optimize_input (simulation, &input[i][0], optimize->
file[i][experiment]);
00210 }
00211 for (; i < MAX_NINPUTS; ++i)
00212     strcpy (&input[i][0], "");
00213 #if DEBUG
00214     fprintf (stderr, "optimize_parse: parsing end\n");
00215 #endif
00216
00217 // Performing the simulation
00218     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00219     buffer2 = g_path_get_dirname (optimize->simulator);
00220     buffer3 = g_path_get_basename (optimize->simulator);
00221     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00222     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
00223             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00224             input[6], input[7], output);
00225     g_free (buffer4);
00226     g_free (buffer3);
00227     g_free (buffer2);
00228 #if DEBUG
00229     fprintf (stderr, "optimize_parse: %s\n", buffer);
00230 #endif
00231     system (buffer);
00232
00233 // Checking the objective value function
00234 if (optimize->evaluator)
00235 {
00236     snprintf (result, 32, "result-%u-%u", simulation, experiment);
00237     buffer2 = g_path_get_dirname (optimize->evaluator);
00238     buffer3 = g_path_get_basename (optimize->evaluator);
00239     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00240     snprintf (buffer, 512, "%s\ " %s %s %s",
00241             buffer4, output, optimize->experiment[experiment], result);
00242     g_free (buffer4);
00243     g_free (buffer3);
00244     g_free (buffer2);
00245 #if DEBUG
00246     fprintf (stderr, "optimize_parse: %s\n", buffer);
00247 #endif
00248     system (buffer);
00249     file_result = g_fopen (result, "r");
00250     e = atof (fgets (buffer, 512, file_result));
00251     fclose (file_result);
00252 }
00253 else
00254 {
00255     strcpy (result, "");
00256     file_result = g_fopen (output, "r");
00257     e = atof (fgets (buffer, 512, file_result));
00258     fclose (file_result);
00259 }
00260
00261 // Removing files
00262 #if !DEBUG
00263 for (i = 0; i < optimize->ninputs; ++i)
00264 {
00265     if (optimize->file[i][0])
00266     {
00267         snprintf (buffer, 512, RM " %s", &input[i][0]);
00268         system (buffer);
00269     }
00270 }
00271     snprintf (buffer, 512, RM " %s %s", output, result);
00272     system (buffer);
00273 #endif
00274
00275 // Processing pending events
00276 show_pending ();
00277
00278 #if DEBUG
00279     fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282 // Returning the objective function
00283 return e * optimize->weight[experiment];
00284 }

```

Here is the call graph for this function:



#### 5.19.2.14 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 433 of file [optimize.c](#).

```

00434 {
00435     unsigned int i;
00436     char buffer[64];
00437     #if DEBUG
00438     fprintf (stderr, "optimize_save_variables: start\n");
00439     #endif
00440     for (i = 0; i < optimize->nvariables; ++i)
00441     {
00442         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00443         fprintf (optimize->file_variables, buffer,
00444             optimize->value[simulation * optimize->
nvariables + i]);
00445     }
00446     fprintf (optimize->file_variables, "%.14le\n", error);
00447     #if DEBUG
00448     fprintf (stderr, "optimize_save_variables: end\n");
00449     #endif
00450 }
  
```

#### 5.19.2.15 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 959 of file [optimize.c](#).

```

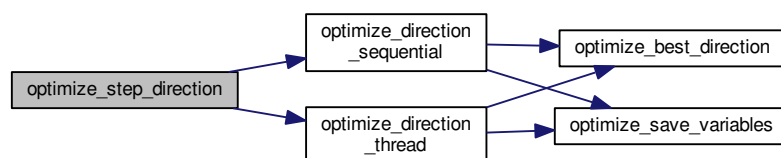
00960 {
00961     GThread *thread[nthreads_direction];
00962     ParallelData data[nthreads_direction];
00963     unsigned int i, j, k, b;
00964     #if DEBUG
00965     fprintf (stderr, "optimize_step_direction: start\n");
00966     #endif
00967     for (i = 0; i < optimize->nestimates; ++i)
00968     {
00969         k = (simulation + i) * optimize->nvariables;
00970         b = optimize->simulation_best[0] * optimize->
  
```

```

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

```

Here is the call graph for this function:



#### 5.19.2.16 void\* optimize\_thread ( ParallelData \* data )

Function to optimize on a thread.

## Parameters

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

## Returns

NULL

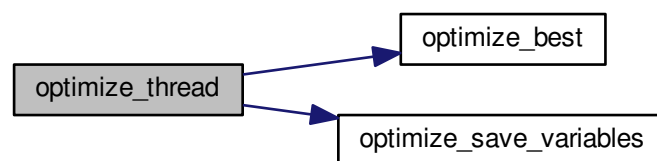
Definition at line 538 of file [optimize.c](#).

```

00539 {
00540     unsigned int i, thread;
00541     double e;
00542     #if DEBUG
00543     fprintf (stderr, "optimize_thread: start\n");
00544     #endif
00545     thread = data->thread;
00546     #if DEBUG
00547     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00548             optimize->thread[thread], optimize->thread[thread + 1]);
00549     #endif
00550     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00551     {
00552         e = optimize_norm (i);
00553         g_mutex_lock (mutex);
00554         optimize_best (i, e);
00555         optimize_save_variables (i, e);
00556         if (e < optimize->thresold)
00557             optimize->stop = 1;
00558         g_mutex_unlock (mutex);
00559         if (optimize->stop)
00560             break;
00561     #if DEBUG
00562     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00563     #endif
00564     }
00565     #if DEBUG
00566     fprintf (stderr, "optimize_thread: end\n");
00567     #endif
00568     g_thread_exit (NULL);
00569     return NULL;
00570 }

```

Here is the call graph for this function:



## 5.20 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 Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,

```

```

00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #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;
00085     unsigned int niterations;
00086     unsigned int nbest;
00087     unsigned int nsaveds;
00088     unsigned int stop;
00089     #if HAVE_MPI
00090     int mpi_rank;
00091     #endif
00092 } Optimize;
00093
00094 typedef struct
00095 {
00096     unsigned int thread;
00097 } 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 extern const xmlChar *result_name;
00138 extern const xmlChar *variables_name;
00139
00140 // Public functions
00141 void optimize_input (unsigned int simulation, char *input,
00142                    GMappedFile * template);
00143 double optimize_parse (unsigned int simulation, unsigned int experiment);
00144 double optimize_norm_euclidian (unsigned int simulation);
00145 double optimize_norm_maximum (unsigned int simulation);
00146 double optimize_norm_p (unsigned int simulation);
00147 double optimize_norm_taxicab (unsigned int simulation);
00148 void optimize_print ();
00149 void optimize_save_variables (unsigned int simulation, double error);
00150 void optimize_best (unsigned int simulation, double value);
00151 void optimize_sequential ();
00152 void *optimize_thread (ParallelData * data);
00153 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00154                    double *error_best);
00155 #if HAVE_MPI
00156 void optimize_synchronise ();
00157 #endif
00158 void optimize_sweep ();
00159 void optimize_MonteCarlo ();
00160 void optimize_best_direction (unsigned int simulation, double value);
00161 void optimize_direction_sequential ();
00162 void *optimize_direction_thread (ParallelData * data);
00163 double optimize_estimate_direction_random (unsigned int variable,
00164                                           unsigned int estimate);
00165 double optimize_estimate_direction_coordinates (unsigned int
00166                                              variable,
00167                                              unsigned int estimate);
00167 void optimize_step_direction (unsigned int simulation);
00168 void optimize_direction ();
00169 double optimize_genetic_objective (Entity * entity);
00170 void optimize_genetic ();
00171 void optimize_save_old ();
00172 void optimize_merge_old ();
00173 void optimize_refine ();
00174 void optimize_step ();
00175 void optimize_iterate ();
00176 void optimize_free ();
00177 void optimize_open ();
00178
00179 #endif

```

## 5.21 utils.c File Reference

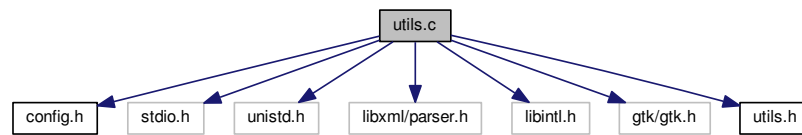
Source file to define some useful functions.

```

#include "config.h"
#include <stdio.h>
#include <unistd.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <gtk/gtk.h>
#include "utils.h"

```

Include dependency graph for utils.c:



## Functions

- void [show\\_pending](#) ()  
*Function to show events on long computation.*
- 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 [cores\\_number](#) ()  
*Function to obtain the cores number.*
- 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.*

### 5.21.1 Detailed Description

Source file to define some useful functions.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.21.2 Function Documentation

#### 5.21.2.1 int cores\_number ( )

Function to obtain the cores number.

#### Returns

Cores number.

Definition at line [328](#) of file [utils.c](#).

```
00329 {
00330     #ifdef G_OS_WIN32
00331         SYSTEM_INFO sysinfo;
00332         GetSystemInfo (&sysinfo);
00333         return sysinfo.dwNumberOfProcessors;
00334     #else
00335         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00336     #endif
00337 }
```

#### 5.21.2.2 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 [352](#) of file [utils.c](#).

```
00353 {
00354     unsigned int i;
00355     for (i = 0; i < n; ++i)
00356         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00357             break;
00358     return i;
00359 }
```

#### 5.21.2.3 void show\_error ( char \* msg )

Function to show a dialog with an error message.



## Parameters

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

Definition at line 108 of file [utils.c](#).

```
00109 {
00110     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00111 }
```

Here is the call graph for this function:



#### 5.21.2.4 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 78 of file [utils.c](#).

```
00079 {
00080     #if HAVE_GTK
00081         GtkMessageDialog *dlg;
00082
00083         // Creating the dialog
00084         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00085             (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00086
00087         // Setting the dialog title
00088         gtk_window_set_title (GTK_WINDOW (dlg), title);
00089
00090         // Showing the dialog and waiting response
00091         gtk_dialog_run (GTK_DIALOG (dlg));
00092
00093         // Closing and freeing memory
00094         gtk_widget_destroy (GTK_WIDGET (dlg));
00095
00096     #else
00097         printf ("%s: %s\n", title, msg);
00098     #endif
00099 }
```

#### 5.21.2.5 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 218 of file [utils.c](#).

```
00219 {
00220     double x = 0.;
00221     xmlChar *buffer;
00222     buffer = xmlGetProp (node, prop);
00223     if (!buffer)
00224         *error_code = 1;
00225     else
00226     {
00227         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00228             *error_code = 2;
00229         else
00230             *error_code = 0;
00231         xmlFree (buffer);
00232     }
00233     return x;
00234 }
```

**5.21.2.6** `double xml_node_get_float_with_default ( xmlDoc * 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.

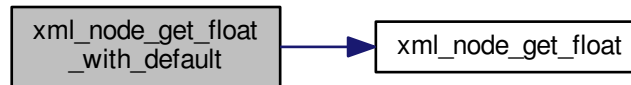
**Returns**

Floating point number value.

Definition at line 252 of file [utils.c](#).

```
00254 {
00255     double x;
00256     if (xmlHasProp (node, prop))
00257         x = xml_node_get_float (node, prop, error_code);
00258     else
00259     {
00260         x = default_value;
00261         *error_code = 0;
00262     }
00263     return x;
00264 }
```

Here is the call graph for this function:



#### 5.21.2.7 int xml\_node\_get\_int ( xmlDoc \* *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 126 of file [utils.c](#).

```

00127 {
00128     int i = 0;
00129     xmlChar *buffer;
00130     buffer = xmlGetProp (node, prop);
00131     if (!buffer)
00132         *error_code = 1;
00133     else
00134     {
00135         if (sscanf ((char *) buffer, "%d", &i) != 1)
00136             *error_code = 2;
00137         else
00138             *error_code = 0;
00139         xmlFree (buffer);
00140     }
00141     return i;
00142 }
  
```

#### 5.21.2.8 int xml\_node\_get\_uint ( xmlDoc \* *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 157 of file [utils.c](#).

```

00158 {
00159     unsigned int i = 0;
00160     xmlChar *buffer;
00161     buffer = xmlGetProp (node, prop);
00162     if (!buffer)
00163         *error_code = 1;
00164     else
00165     {
00166         if (sscanf ((char *) buffer, "%u", &i) != 1)
00167             *error_code = 2;
00168         else
00169             *error_code = 0;
00170         xmlFree (buffer);
00171     }
00172     return i;
00173 }

```

**5.21.2.9** `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.

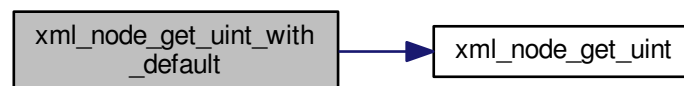
Definition at line 191 of file [utils.c](#).

```

00193 {
00194     unsigned int i;
00195     if (xmlHasProp (node, prop))
00196         i = xml_node_get_uint (node, prop, error_code);
00197     else
00198     {
00199         i = default_value;
00200         *error_code = 0;
00201     }
00202     return i;
00203 }

```

Here is the call graph for this function:



**5.21.2.10** `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 315 of file [utils.c](#).

```

00316 {
00317     xmlChar buffer[64];
00318     snprintf ((char *) buffer, 64, "%.14lg", value);
00319     xmlSetProp (node, prop, buffer);
00320 }
```

#### 5.21.2.11 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 277 of file [utils.c](#).

```

00278 {
00279     xmlChar buffer[64];
00280     snprintf ((char *) buffer, 64, "%d", value);
00281     xmlSetProp (node, prop, buffer);
00282 }
```

#### 5.21.2.12 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 296 of file [utils.c](#).

```

00297 {
00298     xmlChar buffer[64];
00299     snprintf ((char *) buffer, 64, "%u", value);
00300     xmlSetProp (node, prop, buffer);
00301 }
```

## 5.22 utils.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
```

```

00014         this list of conditions and the following disclaimer.
00015
00016         2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <unistd.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #if HAVE_GTK
00039 #include <gtk/gtk.h>
00040 #endif
00041 #include "utils.h"
00042
00043 #if HAVE_GTK
00044 GtkWidget *main_window;
00045 #endif
00046
00047 char *error_message;
00048
00049 void show_pending ()
00050 {
00051     #if HAVE_GTK
00052         while (gtk_events_pending ())
00053             gtk_main_iteration ();
00054     #endif
00055 }
00056
00057 void
00058 show_message (char *title, char *msg, int type)
00059 {
00060     #if HAVE_GTK
00061         GtkMessageDialog *dlg;
00062
00063         // Creating the dialog
00064         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00065             (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00066
00067         // Setting the dialog title
00068         gtk_window_set_title (GTK_WINDOW (dlg), title);
00069
00070         // Showing the dialog and waiting response
00071         gtk_dialog_run (GTK_DIALOG (dlg));
00072
00073         // Closing and freeing memory
00074         gtk_widget_destroy (GTK_WIDGET (dlg));
00075     #else
00076         printf ("%s: %s\n", title, msg);
00077     #endif
00078 }
00079
00080 void
00081 show_error (char *msg)
00082 {
00083     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00084 }
00085
00086 int
00087 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00088 {
00089     int i = 0;
00090     xmlChar *buffer;
00091     buffer = xmlGetProp (node, prop);
00092     if (!buffer)
00093         *error_code = 1;
00094     else
00095     {
00096         if (sscanf ((char *) buffer, "%d", &i) != 1)
00097             *error_code = 2;
00098         else
00099             *error_code = 0;
00100     }
00101 }

```

```

00139     xmlFree (buffer);
00140 }
00141 return i;
00142 }
00143
00156 unsigned int
00157 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00158 {
00159     unsigned int i = 0;
00160     xmlChar *buffer;
00161     buffer = xmlGetProp (node, prop);
00162     if (!buffer)
00163         *error_code = 1;
00164     else
00165     {
00166         if (sscanf ((char *) buffer, "%u", &i) != 1)
00167             *error_code = 2;
00168         else
00169             *error_code = 0;
00170         xmlFree (buffer);
00171     }
00172     return i;
00173 }
00174
00190 unsigned int
00191 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00192                                unsigned int default_value, int *error_code)
00193 {
00194     unsigned int i;
00195     if (xmlHasProp (node, prop))
00196         i = xml_node_get_uint (node, prop, error_code);
00197     else
00198     {
00199         i = default_value;
00200         *error_code = 0;
00201     }
00202     return i;
00203 }
00204
00217 double
00218 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00219 {
00220     double x = 0.;
00221     xmlChar *buffer;
00222     buffer = xmlGetProp (node, prop);
00223     if (!buffer)
00224         *error_code = 1;
00225     else
00226     {
00227         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00228             *error_code = 2;
00229         else
00230             *error_code = 0;
00231         xmlFree (buffer);
00232     }
00233     return x;
00234 }
00235
00251 double
00252 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00253                                  double default_value, int *error_code)
00254 {
00255     double x;
00256     if (xmlHasProp (node, prop))
00257         x = xml_node_get_float (node, prop, error_code);
00258     else
00259     {
00260         x = default_value;
00261         *error_code = 0;
00262     }
00263     return x;
00264 }
00265
00276 void
00277 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00278 {
00279     xmlChar buffer[64];
00280     snprintf ((char *) buffer, 64, "%d", value);
00281     xmlSetProp (node, prop, buffer);
00282 }
00283
00295 void
00296 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00297 {
00298     xmlChar buffer[64];
00299     snprintf ((char *) buffer, 64, "%u", value);
00300     xmlSetProp (node, prop, buffer);

```

```

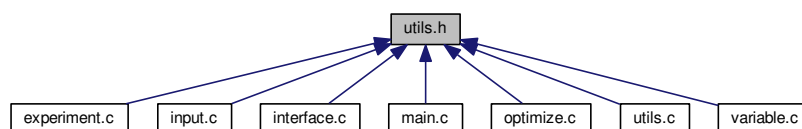
00301 }
00302
00314 void
00315 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00316 {
00317     xmlChar buffer[64];
00318     snprintf ((char *) buffer, 64, "%.14lg", value);
00319     xmlSetProp (node, prop, buffer);
00320 }
00321
00327 int
00328 cores_number ()
00329 {
00330     #ifdef G_OS_WIN32
00331         SYSTEM_INFO sysinfo;
00332         GetSystemInfo (&sysinfo);
00333         return sysinfo.dwNumberOfProcessors;
00334     #else
00335         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00336     #endif
00337 }
00338
00339 #if HAVE_GTK
00340
00351 unsigned int
00352 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00353 {
00354     unsigned int i;
00355     for (i = 0; i < n; ++i)
00356         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00357             break;
00358     return i;
00359 }
00360
00361 #endif

```

## 5.23 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_pending ()`  
Function to show events on long computation.
- `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 [cores\\_number](#) ()

*Function to obtain the cores number.*

- 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.*

### 5.23.1 Detailed Description

Header file to define some useful functions.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.23.2 Function Documentation

#### 5.23.2.1 int [cores\\_number](#) ( )

Function to obtain the cores number.

**Returns**

Cores number.

Definition at line 328 of file [utils.c](#).

```
00329 {
00330 #ifdef G_OS_WIN32
00331     SYSTEM_INFO sysinfo;
00332     GetSystemInfo (&sysinfo);
00333     return sysinfo.dwNumberOfProcessors;
00334 #else
00335     return (int) sysconf (_SC_NPROCESSORS_ONLN);
00336 #endif
00337 }
```

**5.23.2.2 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 352 of file [utils.c](#).

```
00353 {
00354     unsigned int i;
00355     for (i = 0; i < n; ++i)
00356         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00357             break;
00358     return i;
00359 }
```

**5.23.2.3 void show\_error ( char \* *msg* )**

Function to show a dialog with an error message.

**Parameters**

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

Definition at line 108 of file [utils.c](#).

```
00109 {
00110     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00111 }
```

Here is the call graph for this function:



#### 5.23.2.4 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 78 of file [utils.c](#).

```

00079 {
00080 #if HAVE_GTK
00081     GtkMessageDialog *dlg;
00082
00083     // Creating the dialog
00084     dlg = (GtkMessageDialog *) gtk_message_dialog_new
00085         (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00086
00087     // Setting the dialog title
00088     gtk_window_set_title (GTK_WINDOW (dlg), title);
00089
00090     // Showing the dialog and waiting response
00091     gtk_dialog_run (GTK_DIALOG (dlg));
00092
00093     // Closing and freeing memory
00094     gtk_widget_destroy (GTK_WIDGET (dlg));
00095
00096 #else
00097     printf ("%s: %s\n", title, msg);
00098 #endif
00099 }
```

#### 5.23.2.5 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 218 of file [utils.c](#).

```

00219 {
00220     double x = 0.;
00221     xmlChar *buffer;
00222     buffer = xmlGetProp (node, prop);
00223     if (!buffer)
00224         *error_code = 1;
00225     else
00226     {
00227         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00228             *error_code = 2;
00229         else
00230             *error_code = 0;
00231         xmlFree (buffer);
00232     }
00233     return x;
00234 }
```

#### 5.23.2.6 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

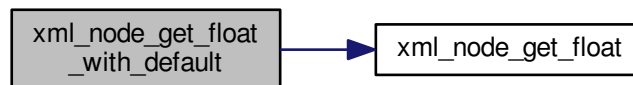
Floating point number value.

Definition at line 252 of file [utils.c](#).

```

00254 {
00255     double x;
00256     if (xmlHasProp (node, prop))
00257         x = xml_node_get_float (node, prop, error_code);
00258     else
00259     {
00260         x = default_value;
00261         *error_code = 0;
00262     }
00263     return x;
00264 }
```

Here is the call graph for this function:



### 5.23.2.7 int xml\_node\_get\_int ( xmlDoc \* 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 126 of file [utils.c](#).

```

00127 {
00128     int i = 0;
00129     xmlChar *buffer;
00130     buffer = xmlGetProp (node, prop);
00131     if (!buffer)
00132         *error_code = 1;
00133     else
00134     {
00135         if (sscanf ((char *) buffer, "%d", &i) != 1)
```

```

00136         *error_code = 2;
00137     else
00138         *error_code = 0;
00139     xmlFree (buffer);
00140 }
00141 return i;
00142 }

```

#### 5.23.2.8 unsigned int xml\_node\_get\_uint ( xmlDoc \* 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 157 of file [utils.c](#).

```

00158 {
00159     unsigned int i = 0;
00160     xmlChar *buffer;
00161     buffer = xmlGetProp (node, prop);
00162     if (!buffer)
00163         *error_code = 1;
00164     else
00165     {
00166         if (sscanf ((char *) buffer, "%u", &i) != 1)
00167             *error_code = 2;
00168         else
00169             *error_code = 0;
00170         xmlFree (buffer);
00171     }
00172     return i;
00173 }

```

#### 5.23.2.9 unsigned int xml\_node\_get\_uint\_with\_default ( xmlDoc \* 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 191 of file [utils.c](#).

```

00193 {
00194     unsigned int i;
00195     if (xmlHasProp (node, prop))
00196         i = xml_node_get_uint (node, prop, error_code);
00197     else
00198     {
00199         i = default_value;

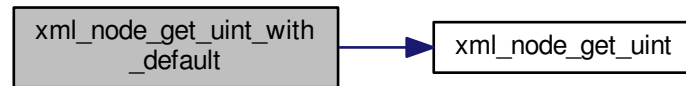
```

```

00200     *error_code = 0;
00201     }
00202     return i;
00203 }

```

Here is the call graph for this function:



#### 5.23.2.10 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 315 of file [utils.c](#).

```

00316 {
00317     xmlChar buffer[64];
00318     snprintf ((char *) buffer, 64, "%.14lg", value);
00319     xmlSetProp (node, prop, buffer);
00320 }

```

#### 5.23.2.11 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 277 of file [utils.c](#).

```

00278 {
00279     xmlChar buffer[64];
00280     snprintf ((char *) buffer, 64, "%d", value);
00281     xmlSetProp (node, prop, buffer);
00282 }

```

#### 5.23.2.12 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 296 of file [utils.c](#).

```
00297 {
00298     xmlChar buffer[64];
00299     snprintf ((char *) buffer, 64, "%u", value);
00300     xmlSetProp (node, prop, buffer);
00301 }
```

## 5.24 utils.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef UTILS__H
00033 #define UTILS__H 1
00034
00035 #if HAVE_GTK
00036 #define ERROR_TYPE GTK_MESSAGE_ERROR
00037 #define INFO_TYPE GTK_MESSAGE_INFO
00038 extern GtkWidget *main_window;
00039 #else
00040 #define ERROR_TYPE 0
00041 #define INFO_TYPE 0
00042 #endif
00043
00044 extern char *error_message;
00045
00046 // Public functions
00047 void show_pending ();
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);
```



```

00076 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00077 int cores_number ();
00078 #if HAVE_GTK
00079 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00080 #endif
00081
00082 #endif

```

## 5.25 variable.c File Reference

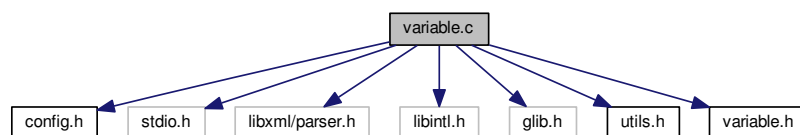
Source file to define the variable data.

```

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

```

Include dependency graph for variable.c:



### Macros

- `#define _GNU_SOURCE`
- `#define DEBUG 0`

*Macro to debug.*

### Functions

- void `variable_new` (`Variable *variable`)  
*Function to create a new `Variable` struct.*
- void `variable_free` (`Variable *variable`)  
*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` (`Variable *variable`, `xmlNode *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.*

### 5.25.1 Detailed Description

Source file to define the variable data.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.25.2 Function Documentation

#### 5.25.2.1 void variable\_error ( Variable \* variable, char \* message )

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

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>message</i>	Error message.

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

```

00105 {
00106     char buffer[64];
00107     if (!variable->name)
00108         snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00109     else
00110         snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00111                 message);
00112     error_message = g_strdup (buffer);
00113 }
```

#### 5.25.2.2 void variable\_free ( Variable \* variable )

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

##### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
-----------------	----------------------------------

Definition at line 84 of file [variable.c](#).

```

00085 {
00086     #if DEBUG
00087         fprintf (stderr, "variable_free: start\n");
00088     #endif
00089     xmlFree (variable->name);
00090     #if DEBUG
00091         fprintf (stderr, "variable_free: end\n");
00092     #endif
00093 }
```

#### 5.25.2.3 void variable\_new ( Variable \* variable )

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

## Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
-----------------	----------------------------------

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

```

00067 {
00068     #if DEBUG
00069         fprintf (stderr, "variable_new: start\n");
00070     #endif
00071     variable->name = NULL;
00072     #if DEBUG
00073         fprintf (stderr, "variable_new: end\n");
00074     #endif
00075 }
```

#### 5.25.2.4 int variable\_open ( Variable \* variable, xmlNode \* node, unsigned int algorithm, unsigned int nsteps )

Function to open the variable file.

## Parameters

<i>variable</i>	<a href="#">Variable</a> 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 130 of file [variable.c](#).

```

00132 {
00133     char *msg;
00134     int error_code;
00135
00136     #if DEBUG
00137         fprintf (stderr, "variable_open: start\n");
00138     #endif
00139
00140     variable->name = (char*) xmlGetProp (node, XML_NAME);
00141     if (!variable->name)
00142     {
00143         variable_error (variable, gettext ("no name"));
00144         goto exit_on_error;
00145     }
00146     if (xmlHasProp (node, XML_MINIMUM))
00147     {
00148         variable->rangemin = xml_node_get_float (node,
XML_MINIMUM, &error_code);
00149         if (error_code)
00150         {
00151             variable_error (variable, gettext ("bad minimum"));
00152             goto exit_on_error;
00153         }
00154         variable->rangeminabs
= xml_node_get_float_with_default (node,
XML_ABSOLUTE_MINIMUM,
00156                                     -G_MAXDOUBLE, &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad absolute minimum"));
00160             goto exit_on_error;
00161         }
00162         if (variable->rangemin < variable->rangeminabs)
00163         {
00164             variable_error (variable, gettext ("minimum range not allowed"));
00165             goto exit_on_error;
00166         }
00167     }
00168     else
00169     {
00170         variable_error (variable, gettext ("no minimum range"));
00171         goto exit_on_error;

```

```

00172     }
00173     if (xmlHasProp (node, XML_MAXIMUM))
00174     {
00175         variable->rangemax
00176         = xml_node_get_float (node, XML_MAXIMUM, &error_code);
00177         if (error_code)
00178         {
00179             variable_error (variable, gettext ("bad maximum"));
00180             goto exit_on_error;
00181         }
00182         variable->rangemaxabs
00183         = xml_node_get_float_with_default (node,
XML_ABSOLUTE_MAXIMUM,
00184                                           G_MAXDOUBLE, &error_code);
00185         if (error_code)
00186         {
00187             variable_error (variable, gettext ("bad absolute maximum"));
00188             goto exit_on_error;
00189         }
00190         if (variable->rangemax > variable->rangemaxabs)
00191         {
00192             variable_error (variable, gettext ("maximum range not allowed"));
00193             goto exit_on_error;
00194         }
00195         if (variable->rangemax < variable->rangemin)
00196         {
00197             variable_error (variable, gettext ("bad range"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemaxabs < variable->rangeminabs)
00201         {
00202             variable_error (variable, gettext ("bad absolute range"));
00203             goto exit_on_error;
00204         }
00205     }
00206     else
00207     {
00208         variable_error (variable, gettext ("no maximum range"));
00209         goto exit_on_error;
00210     }
00211     variable->precision
00212     = xml_node_get_uint_with_default (node,
XML_PRECISION,
00213                                     DEFAULT_PRECISION, &error_code);
00214     if (error_code || variable->precision >= NPRECISIONS)
00215     {
00216         variable_error (variable, gettext ("bad precision"));
00217         goto exit_on_error;
00218     }
00219     if (algorithm == ALGORITHM_SWEEP)
00220     {
00221         if (xmlHasProp (node, XML_NSWEEPS))
00222         {
00223             variable->nsweeps
00224             = xml_node_get_uint (node, XML_NSWEEPS, &error_code);
00225             if (error_code || !variable->nsweeps)
00226             {
00227                 variable_error (variable, gettext ("bad sweeps"));
00228                 goto exit_on_error;
00229             }
00230         }
00231         else
00232         {
00233             variable_error (variable, gettext ("no sweeps number"));
00234             goto exit_on_error;
00235         }
00236         #if DEBUG
00237         fprintf (stderr, "variable_open: nsweeps=%u\n", variable->nsweeps);
00238         #endif
00239     }
00240     if (algorithm == ALGORITHM_GENETIC)
00241     {
00242         // Obtaining bits representing each variable
00243         if (xmlHasProp (node, XML_NBITS))
00244         {
00245             variable->nbits = xml_node_get_uint (node,
XML_NBITS,
00246                                                 &error_code);
00247             if (error_code || !variable->nbits)
00248             {
00249                 variable_error (variable, gettext ("invalid bits number"));
00250                 goto exit_on_error;
00251             }
00252         }
00253         else
00254         {
00255             variable_error (variable, gettext ("no bits number"));
00256             goto exit_on_error;
00257         }
00258     }

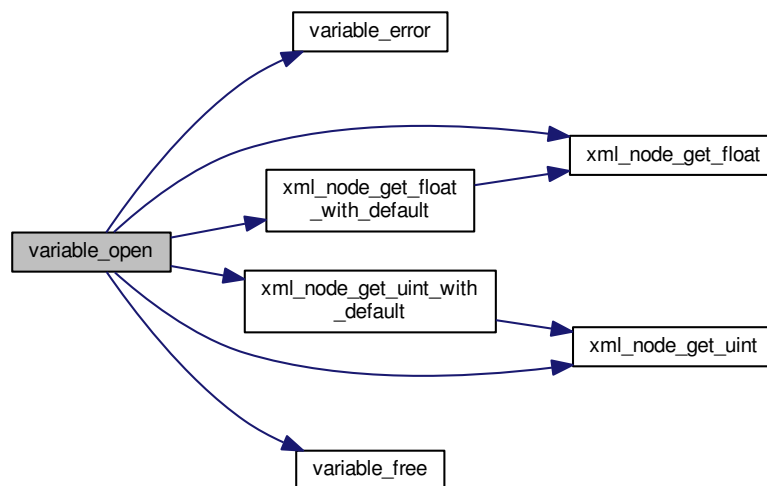
```

```

00256     }
00257 }
00258 else if (nsteps)
00259 {
00260     variable->step
00261     = xml_node_get_float (node, XML_STEP, &error_code);
00262     if (error_code || variable->step < 0.)
00263     {
00264         variable_error (variable, gettext ("bad step size"));
00265         goto exit_on_error;
00266     }
00267 }
00268
00269 #if DEBUG
00270 fprintf (stderr, "variable_open: end\n");
00271 #endif
00272 return 1;
00273
00274 exit_on_error:
00275     variable_free (variable);
00276 #if DEBUG
00277 fprintf (stderr, "variable_open: end\n");
00278 #endif
00279 return 0;
00280 }

```

Here is the call graph for this function:



## 5.25.3 Variable Documentation

### 5.25.3.1 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 49 of file [variable.c](#).

### 5.25.3.2 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 54 of file [variable.c](#).

## 5.26 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 Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include "utils.h"
00039 #include "variable.h"
00040
00041 #define DEBUG 0
00042
00043 const char *format[NPRECISIONS] = {
00044     "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00045     "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00046 };
00047
00048 const double precision[NPRECISIONS] = {
00049     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,
00050     1e-13, 1e-14
00051 };
00052
00053 void
00054 variable_new (Variable * variable)
00055 {
00056     #if DEBUG
00057         fprintf (stderr, "variable_new: start\n");
00058     #endif
00059     variable->name = NULL;
00060     #if DEBUG
00061         fprintf (stderr, "variable_new: end\n");
00062     #endif
00063 }
00064
00065 void
```

```

00084 variable_free (Variable * variable)
00085 {
00086     #if DEBUG
00087         fprintf (stderr, "variable_free: start\n");
00088     #endif
00089     xmlFree (variable->name);
00090     #if DEBUG
00091         fprintf (stderr, "variable_free: end\n");
00092     #endif
00093 }
00094
00103 void
00104 variable_error (Variable * variable, char *message)
00105 {
00106     char buffer[64];
00107     if (!variable->name)
00108         snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00109     else
00110         snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00111                 message);
00112     error_message = g_strdup (buffer);
00113 }
00114
00129 int
00130 variable_open (Variable * variable, xmlNode * node, unsigned int algorithm,
00131               unsigned int nsteps)
00132 {
00133     char *msg;
00134     int error_code;
00135
00136     #if DEBUG
00137         fprintf (stderr, "variable_open: start\n");
00138     #endif
00139
00140     variable->name = (char*) xmlGetProp (node, XML_NAME);
00141     if (!variable->name)
00142     {
00143         variable_error (variable, gettext ("no name"));
00144         goto exit_on_error;
00145     }
00146     if (xmlHasProp (node, XML_MINIMUM))
00147     {
00148         variable->rangemin = xml_node_get_float (node,
00149 XML_MINIMUM, &error_code);
00150         if (error_code)
00151         {
00152             variable_error (variable, gettext ("bad minimum"));
00153             goto exit_on_error;
00154         }
00155         variable->rangeminabs
00156         = xml_node_get_float_with_default (node,
00157 XML_ABSOLUTE_MINIMUM,
00158                                           -G_MAXDOUBLE, &error_code);
00159         if (error_code)
00160         {
00161             variable_error (variable, gettext ("bad absolute minimum"));
00162             goto exit_on_error;
00163         }
00164         if (variable->rangemin < variable->rangeminabs)
00165         {
00166             variable_error (variable, gettext ("minimum range not allowed"));
00167             goto exit_on_error;
00168         }
00169     }
00170     else
00171     {
00172         variable_error (variable, gettext ("no minimum range"));
00173         goto exit_on_error;
00174     }
00175     if (xmlHasProp (node, XML_MAXIMUM))
00176     {
00177         variable->rangemax
00178         = xml_node_get_float (node, XML_MAXIMUM, &error_code);
00179         if (error_code)
00180         {
00181             variable_error (variable, gettext ("bad maximum"));
00182             goto exit_on_error;
00183         }
00184         variable->rangemaxabs
00185         = xml_node_get_float_with_default (node,
00186 XML_ABSOLUTE_MAXIMUM,
00187                                           G_MAXDOUBLE, &error_code);
00188         if (error_code)
00189         {
00190             variable_error (variable, gettext ("bad absolute maximum"));
00191             goto exit_on_error;
00192         }
00193     }
00194     else
00195     {
00196         variable_error (variable, gettext ("no maximum range"));
00197         goto exit_on_error;
00198     }
00199 }

```

```

00190     if (variable->rangemax > variable->rangemaxabs)
00191     {
00192         variable_error (variable, gettext ("maximum range not allowed"));
00193         goto exit_on_error;
00194     }
00195     if (variable->rangemax < variable->rangemin)
00196     {
00197         variable_error (variable, gettext ("bad range"));
00198         goto exit_on_error;
00199     }
00200     if (variable->rangemaxabs < variable->rangeminabs)
00201     {
00202         variable_error (variable, gettext ("bad absolute range"));
00203         goto exit_on_error;
00204     }
00205 }
00206 else
00207 {
00208     variable_error (variable, gettext ("no maximum range"));
00209     goto exit_on_error;
00210 }
00211 variable->precision
00212 = xml_node_get_uint_with_default (node,
XML_PRECISION,
                                DEFAULT_PRECISION, &error_code);
00213 if (error_code || variable->precision >= NPRECISIONS)
00214 {
00215     variable_error (variable, gettext ("bad precision"));
00216     goto exit_on_error;
00217 }
00218 }
00219 if (algorithm == ALGORITHM_SWEEP)
00220 {
00221     if (xmlHasProp (node, XML_NSWEEPS))
00222     {
00223         variable->nsweeps
00224         = xml_node_get_uint (node, XML_NSWEEPS, &error_code);
00225         if (error_code || !variable->nsweeps)
00226         {
00227             variable_error (variable, gettext ("bad sweeps"));
00228             goto exit_on_error;
00229         }
00230     }
00231     else
00232     {
00233         variable_error (variable, gettext ("no sweeps number"));
00234         goto exit_on_error;
00235     }
00236 #if DEBUG
00237     fprintf (stderr, "variable_open: nsweeps=%u\n", variable->nsweeps);
00238 #endif
00239 }
00240 if (algorithm == ALGORITHM_GENETIC)
00241 {
00242     // Obtaining bits representing each variable
00243     if (xmlHasProp (node, XML_NBITS))
00244     {
00245         variable->nbits = xml_node_get_uint (node,
XML_NBITS, &error_code);
00246         if (error_code || !variable->nbits)
00247         {
00248             variable_error (variable, gettext ("invalid bits number"));
00249             goto exit_on_error;
00250         }
00251     }
00252     else
00253     {
00254         variable_error (variable, gettext ("no bits number"));
00255         goto exit_on_error;
00256     }
00257 }
00258 else if (nsteps)
00259 {
00260     variable->step
00261     = xml_node_get_float (node, XML_STEP, &error_code);
00262     if (error_code || variable->step < 0.)
00263     {
00264         variable_error (variable, gettext ("bad step size"));
00265         goto exit_on_error;
00266     }
00267 }
00268 #if DEBUG
00269     fprintf (stderr, "variable_open: end\n");
00270 #endif
00271 return 1;
00272
00273
00274 exit_on_error:

```



```

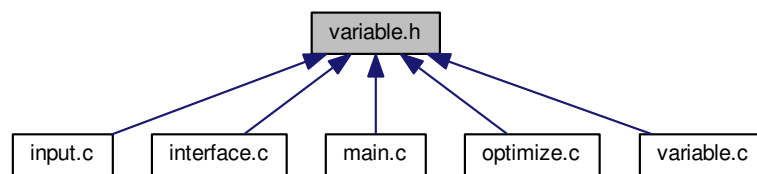
00275     variable_free (variable);
00276     #if DEBUG
00277     fprintf (stderr, "variable_open: end\n");
00278     #endif
00279     return 0;
00280 }

```

## 5.27 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)  
*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](#) ([Variable](#) \*variable, xmlDoc \*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.*

### 5.27.1 Detailed Description

Header file to define the variable data.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.27.2 Enumeration Type Documentation

#### 5.27.2.1 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 };
```

### 5.27.3 Function Documentation

#### 5.27.3.1 void variable\_error ( Variable \* variable, char \* message )

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

#### Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
<i>message</i>	Error message.

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

```
00105 {
00106     char buffer[64];
00107     if (!variable->name)
00108         snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00109     else
00110         snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00111                 message);
00112     error_message = g_strdup (buffer);
00113 }
```

#### 5.27.3.2 void variable\_free ( Variable \* variable )

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

## Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
-----------------	----------------------------------

Definition at line 84 of file [variable.c](#).

```

00085 {
00086     #if DEBUG
00087         fprintf (stderr, "variable_free: start\n");
00088     #endif
00089     xmlFree (variable->name);
00090     #if DEBUG
00091         fprintf (stderr, "variable_free: end\n");
00092     #endif
00093 }
```

5.27.3.3 void variable\_new ( [Variable](#) \* *variable* )

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

## Parameters

<i>variable</i>	<a href="#">Variable</a> struct.
-----------------	----------------------------------

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

```

00067 {
00068     #if DEBUG
00069         fprintf (stderr, "variable_new: start\n");
00070     #endif
00071     variable->name = NULL;
00072     #if DEBUG
00073         fprintf (stderr, "variable_new: end\n");
00074     #endif
00075 }
```

5.27.3.4 int variable\_open ( [Variable](#) \* *variable*, [xmlNode](#) \* *node*, unsigned int *algorithm*, unsigned int *nsteps* )

Function to open the variable file.

## Parameters

<i>variable</i>	<a href="#">Variable</a> 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 130 of file [variable.c](#).

```

00132 {
00133     char *msg;
00134     int error_code;
00135
00136     #if DEBUG
00137         fprintf (stderr, "variable_open: start\n");
00138     #endif
00139
00140     variable->name = (char*) xmlGetProp (node, XML_NAME);
00141     if (!variable->name)
00142     {
00143         variable_error (variable, gettext ("no name"));
00144         goto exit_on_error;
00145     }
00146     if (xmlHasProp (node, XML_MINIMUM))
```

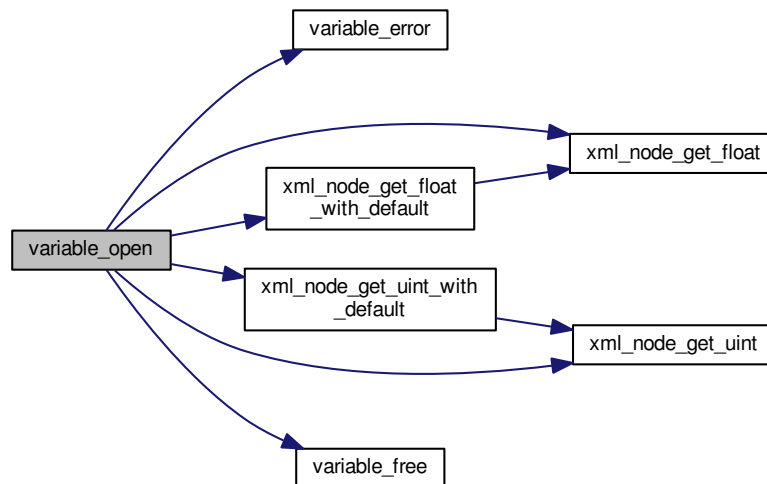
```

00147     {
00148         variable->rangemin = xml_node_get_float (node,
XML_MINIMUM, &error_code);
00149         if (error_code)
00150         {
00151             variable_error (variable, gettext ("bad minimum"));
00152             goto exit_on_error;
00153         }
00154         variable->rangeminabs
00155         = xml_node_get_float_with_default (node,
XML_ABSOLUTE_MINIMUM,
00156                                           -G_MAXDOUBLE, &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad absolute minimum"));
00160             goto exit_on_error;
00161         }
00162         if (variable->rangemin < variable->rangeminabs)
00163         {
00164             variable_error (variable, gettext ("minimum range not allowed"));
00165             goto exit_on_error;
00166         }
00167     }
00168     else
00169     {
00170         variable_error (variable, gettext ("no minimum range"));
00171         goto exit_on_error;
00172     }
00173     if (xmlHasProp (node, XML_MAXIMUM))
00174     {
00175         variable->rangemax
00176         = xml_node_get_float (node, XML_MAXIMUM, &error_code);
00177         if (error_code)
00178         {
00179             variable_error (variable, gettext ("bad maximum"));
00180             goto exit_on_error;
00181         }
00182         variable->rangemaxabs
00183         = xml_node_get_float_with_default (node,
XML_ABSOLUTE_MAXIMUM,
00184                                           G_MAXDOUBLE, &error_code);
00185         if (error_code)
00186         {
00187             variable_error (variable, gettext ("bad absolute maximum"));
00188             goto exit_on_error;
00189         }
00190         if (variable->rangemax > variable->rangemaxabs)
00191         {
00192             variable_error (variable, gettext ("maximum range not allowed"));
00193             goto exit_on_error;
00194         }
00195         if (variable->rangemax < variable->rangemin)
00196         {
00197             variable_error (variable, gettext ("bad range"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemaxabs < variable->rangeminabs)
00201         {
00202             variable_error (variable, gettext ("bad absolute range"));
00203             goto exit_on_error;
00204         }
00205     }
00206     else
00207     {
00208         variable_error (variable, gettext ("no maximum range"));
00209         goto exit_on_error;
00210     }
00211     variable->precision
00212     = xml_node_get_uint_with_default (node,
XML_PRECISION,
00213                                     DEFAULT_PRECISION, &error_code);
00214     if (error_code || variable->precision >= NPRECISIONS)
00215     {
00216         variable_error (variable, gettext ("bad precision"));
00217         goto exit_on_error;
00218     }
00219     if (algorithm == ALGORITHM_SWEEP)
00220     {
00221         if (xmlHasProp (node, XML_NSWEEPS))
00222         {
00223             variable->nsweeps
00224             = xml_node_get_uint (node, XML_NSWEEPS, &error_code);
00225             if (error_code || !variable->nsweeps)
00226             {
00227                 variable_error (variable, gettext ("bad sweeps"));
00228                 goto exit_on_error;
00229             }

```

```
00230     }
00231     else
00232     {
00233         variable_error (variable, gettext ("no sweeps number"));
00234         goto exit_on_error;
00235     }
00236 #if DEBUG
00237     fprintf (stderr, "variable_open: nsweeps=%u\n", variable->nsweeps);
00238 #endif
00239 }
00240 if (algorithm == ALGORITHM_GENETIC)
00241 {
00242     // Obtaining bits representing each variable
00243     if (xmlHasProp (node, XML_NBITS))
00244     {
00245         variable->nbits = xml_node_get_uint (node,
XML_NBITS, &error_code);
00246         if (error_code || !variable->nbits)
00247         {
00248             variable_error (variable, gettext ("invalid bits number"));
00249             goto exit_on_error;
00250         }
00251     }
00252     else
00253     {
00254         variable_error (variable, gettext ("no bits number"));
00255         goto exit_on_error;
00256     }
00257 }
00258 else if (nsteps)
00259 {
00260     variable->step
00261     = xml_node_get_float (node, XML_STEP, &error_code);
00262     if (error_code || variable->step < 0.)
00263     {
00264         variable_error (variable, gettext ("bad step size"));
00265         goto exit_on_error;
00266     }
00267 }
00268
00269 #if DEBUG
00270     fprintf (stderr, "variable_open: end\n");
00271 #endif
00272     return 1;
00273
00274 exit_on_error:
00275     variable_free (variable);
00276 #if DEBUG
00277     fprintf (stderr, "variable_open: end\n");
00278 #endif
00279     return 0;
00280 }
```

Here is the call graph for this function:



## 5.28 variable.h

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2016, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032  #ifndef VARIABLE__H
00033  #define VARIABLE__H 1
00034
00035  enum Algorithm
00036  {
00037      ALGORITHM_MONTE_CARLO = 0,
00038      ALGORITHM_SWEEP = 1,
00039      ALGORITHM_GENETIC = 2
00040  };
00041
00042  typedef struct
00043  {
00044      char *name;
00045      double rangemin;
00046      double rangemax;
  
```

```
00061 double rangeminabs;
00062 double rangemaxabs;
00063 double step;
00064 unsigned int precision;
00065 unsigned int nsweeps;
00066 unsigned int nbits;
00067 } Variable;
00068
00069 extern const char *format[NPRECISIONS];
00070 extern const double precision[NPRECISIONS];
00071
00072 // Public functions
00073 void variable_new (Variable * variable);
00074 void variable_free (Variable * variable);
00075 void variable_error (Variable * variable, char *message);
00076 int variable_open (Variable * variable, xmlNode * node, unsigned int algorithm,
00077                  unsigned int nsteps);
00078
00079 #endif
00080
```





# Index

ALGORITHM\_GENETIC  
    variable.h, [210](#)  
ALGORITHM\_MONTE\_CARLO  
    variable.h, [210](#)  
ALGORITHM\_SWEEP  
    variable.h, [210](#)  
Algorithm  
    variable.h, [210](#)  
  
config.h, [27](#)  
cores\_number  
    utils.c, [184](#)  
    utils.h, [193](#)  
  
DIRECTION\_METHOD\_COORDINATES  
    input.h, [58](#)  
DIRECTION\_METHOD\_RANDOM  
    input.h, [58](#)  
DirectionMethod  
    input.h, [58](#)  
  
ERROR\_NORM\_EUCLIDIAN  
    input.h, [58](#)  
ERROR\_NORM\_MAXIMUM  
    input.h, [58](#)  
ERROR\_NORM\_P  
    input.h, [58](#)  
ERROR\_NORM\_TAXICAB  
    input.h, [58](#)  
ErrorNorm  
    input.h, [58](#)  
Experiment, [13](#)  
experiment.c, [31](#)  
    experiment\_error, [32](#)  
    experiment\_free, [32](#)  
    experiment\_new, [34](#)  
    experiment\_open, [34](#)  
    template, [36](#)  
experiment.h, [38](#)  
    experiment\_error, [39](#)  
    experiment\_free, [40](#)  
    experiment\_new, [40](#)  
    experiment\_open, [40](#)  
experiment\_error  
    experiment.c, [32](#)  
    experiment.h, [39](#)  
experiment\_free  
    experiment.c, [32](#)  
    experiment.h, [40](#)  
experiment\_new  
    experiment.c, [34](#)  
    experiment.h, [40](#)  
  
format  
    variable.c, [205](#)  
  
gtk\_array\_get\_active  
    interface.h, [107](#)  
    utils.c, [184](#)  
    utils.h, [194](#)  
  
Input, [13](#)  
input.c, [43](#)  
    input\_error, [44](#)  
    input\_open, [44](#)  
input.h, [56](#)  
    DIRECTION\_METHOD\_COORDINATES, [58](#)  
    DIRECTION\_METHOD\_RANDOM, [58](#)  
    DirectionMethod, [58](#)  
    ERROR\_NORM\_EUCLIDIAN, [58](#)  
    ERROR\_NORM\_MAXIMUM, [58](#)  
    ERROR\_NORM\_P, [58](#)  
    ERROR\_NORM\_TAXICAB, [58](#)  
    ErrorNorm, [58](#)  
    input\_error, [58](#)  
    input\_open, [58](#)  
input\_error  
    input.c, [44](#)  
    input.h, [58](#)  
input\_open  
    input.c, [44](#)  
    input.h, [58](#)  
input\_save  
    interface.c, [68](#)  
    interface.h, [108](#)  
input\_save\_direction  
    interface.c, [70](#)  
interface.c, [65](#)  
    input\_save, [68](#)  
    input\_save\_direction, [70](#)  
    window\_get\_algorithm, [71](#)  
    window\_get\_direction, [72](#)  
    window\_get\_norm, [72](#)  
    window\_read, [73](#)  
    window\_save, [75](#)  
    window\_template\_experiment, [77](#)  
interface.h, [105](#)

- gtk\_array\_get\_active, 107
- input\_save, 108
- window\_get\_algorithm, 110
- window\_get\_direction, 111
- window\_get\_norm, 111
- window\_read, 112
- window\_save, 114
- window\_template\_experiment, 116
- main
  - main.c, 120
- main.c, 119
  - main, 120
- Optimize, 15
  - thread\_direction, 18
- optimize.c, 126
  - optimize\_best, 129
  - optimize\_best\_direction, 130
  - optimize\_direction\_sequential, 130
  - optimize\_direction\_thread, 132
  - optimize\_estimate\_direction\_coordinates, 133
  - optimize\_estimate\_direction\_random, 134
  - optimize\_genetic\_objective, 134
  - optimize\_input, 135
  - optimize\_merge, 136
  - optimize\_norm\_euclidian, 136
  - optimize\_norm\_maximum, 138
  - optimize\_norm\_p, 139
  - optimize\_norm\_taxicab, 140
  - optimize\_parse, 140
  - optimize\_save\_variables, 142
  - optimize\_step\_direction, 142
  - optimize\_thread, 144
- optimize.h, 162
  - optimize\_best, 165
  - optimize\_best\_direction, 166
  - optimize\_direction\_thread, 166
  - optimize\_estimate\_direction\_coordinates, 167
  - optimize\_estimate\_direction\_random, 167
  - optimize\_genetic\_objective, 169
  - optimize\_input, 169
  - optimize\_merge, 170
  - optimize\_norm\_euclidian, 172
  - optimize\_norm\_maximum, 173
  - optimize\_norm\_p, 174
  - optimize\_norm\_taxicab, 174
  - optimize\_parse, 176
  - optimize\_save\_variables, 178
  - optimize\_step\_direction, 178
  - optimize\_thread, 179
- optimize\_best
  - optimize.c, 129
  - optimize.h, 165
- optimize\_best\_direction
  - optimize.c, 130
  - optimize.h, 166
- optimize\_direction\_sequential
  - optimize.c, 130
- optimize\_direction\_thread
  - optimize.c, 132
  - optimize.h, 166
- optimize\_estimate\_direction\_coordinates
  - optimize.c, 133
  - optimize.h, 167
- optimize\_estimate\_direction\_random
  - optimize.c, 134
  - optimize.h, 167
- optimize\_genetic\_objective
  - optimize.c, 134
  - optimize.h, 169
- optimize\_input
  - optimize.c, 135
  - optimize.h, 169
- optimize\_merge
  - optimize.c, 136
  - optimize.h, 170
- optimize\_norm\_euclidian
  - optimize.c, 136
  - optimize.h, 172
- optimize\_norm\_maximum
  - optimize.c, 138
  - optimize.h, 173
- optimize\_norm\_p
  - optimize.c, 139
  - optimize.h, 174
- optimize\_norm\_taxicab
  - optimize.c, 140
  - optimize.h, 174
- optimize\_parse
  - optimize.c, 140
  - optimize.h, 176
- optimize\_save\_variables
  - optimize.c, 142
  - optimize.h, 178
- optimize\_step\_direction
  - optimize.c, 142
  - optimize.h, 178
- optimize\_thread
  - optimize.c, 144
  - optimize.h, 179
- Options, 18
- ParallelData, 18
- precision
  - variable.c, 205
- Running, 19
- show\_error
  - utils.c, 184
  - utils.h, 194
- show\_message
  - utils.c, 185
  - utils.h, 194
- template
  - experiment.c, 36

- thread\_direction
  - Optimize, 18
- utils.c, 182
  - cores\_number, 184
  - gtk\_array\_get\_active, 184
  - show\_error, 184
  - show\_message, 185
  - xml\_node\_get\_float, 185
  - xml\_node\_get\_float\_with\_default, 186
  - xml\_node\_get\_int, 187
  - xml\_node\_get\_uint, 187
  - xml\_node\_get\_uint\_with\_default, 188
  - xml\_node\_set\_float, 188
  - xml\_node\_set\_int, 189
  - xml\_node\_set\_uint, 189
- utils.h, 192
  - cores\_number, 193
  - gtk\_array\_get\_active, 194
  - show\_error, 194
  - show\_message, 194
  - xml\_node\_get\_float, 196
  - xml\_node\_get\_float\_with\_default, 196
  - xml\_node\_get\_int, 197
  - xml\_node\_get\_uint, 198
  - xml\_node\_get\_uint\_with\_default, 198
  - xml\_node\_set\_float, 199
  - xml\_node\_set\_int, 199
  - xml\_node\_set\_uint, 199
- Variable, 19
- variable.c, 201
  - format, 205
  - precision, 205
  - variable\_error, 202
  - variable\_free, 202
  - variable\_new, 202
  - variable\_open, 203
- variable.h, 209
  - ALGORITHM\_GENETIC, 210
  - ALGORITHM\_MONTE\_CARLO, 210
  - ALGORITHM\_SWEEP, 210
  - Algorithm, 210
  - variable\_error, 210
  - variable\_free, 210
  - variable\_new, 211
  - variable\_open, 211
- variable\_error
  - variable.c, 202
  - variable.h, 210
- variable\_free
  - variable.c, 202
  - variable.h, 210
- variable\_new
  - variable.c, 202
  - variable.h, 211
- variable\_open
  - variable.c, 203
  - variable.h, 211
- Window, 20
- window\_get\_algorithm
  - interface.c, 71
  - interface.h, 110
- window\_get\_direction
  - interface.c, 72
  - interface.h, 111
- window\_get\_norm
  - interface.c, 72
  - interface.h, 111
- window\_read
  - interface.c, 73
  - interface.h, 112
- window\_save
  - interface.c, 75
  - interface.h, 114
- window\_template\_experiment
  - interface.c, 77
  - interface.h, 116
- xml\_node\_get\_float
  - utils.c, 185
  - utils.h, 196
- xml\_node\_get\_float\_with\_default
  - utils.c, 186
  - utils.h, 196
- xml\_node\_get\_int
  - utils.c, 187
  - utils.h, 197
- xml\_node\_get\_uint
  - utils.c, 187
  - utils.h, 198
- xml\_node\_get\_uint\_with\_default
  - utils.c, 188
  - utils.h, 198
- xml\_node\_set\_float
  - utils.c, 188
  - utils.h, 199
- xml\_node\_set\_int
  - utils.c, 189
  - utils.h, 199
- xml\_node\_set\_uint
  - utils.c, 189
  - utils.h, 199